Reaaliaikaisten internet-pelien verkkotekniikasta

Aku Häsänen

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Timo Poranen
Helmikuu 2014

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
HÄSÄNEN, AKU: Reaaliaikaisten internet-pelien verkkotekniikasta
Pro gradu -tutkielma, 75 sivua, 30 liitesivua
Helmikuu 2014

Tämä pro gradu -tutkielma käsittelee reaaliaikaisissa internet-peleissä käytettäviä sovelluskerroksen verkkoprotokollia. Tutkielmassa selvitetään, millaisia erityisvaatimuksia reaaliaikainen pelikokemus asettaa pelin verkkokoodille, ja kuinka nämä vaatimukset on yritetty täyttää tutkimukseen valitussa esimerkkitapauksessa. Tutkimuksen kohteena on ensisijaisesti pelin avoimen lähdekoodin sekä vakiintuneen aseman ansiosta Id Softwaren Quake 3, jonka verkkoteknisiä ratkaisuja voitaneen pitää eräänlaisena merkkipaaluna reaaliaikaisten verkkopelien historiassa. Tutkimuksen tuloksena on yksityiskohtainen ja laaja-alainen kokonaiskuva reaaliaikaisen internet-pelin verkkoprotokollan toteuttamiseen liittyvistä haasteista ja ratkaisuista sekä ajatuksia mahdollisista kehityssuuntauksista ja niiden toteuttamisesta tutkielmassa käsitellyn verkkoprotokollan tarjoamalle pohjalle.

Avainsanat ja -sanonnat: verkko-ohjelmointi, peliohjelmointi, C-ohjelmointi, verkkoprotokollat

Sisällysluettelo

1. Johdanto....1
2. Verkkoyhteyksistä....3
2.1. Verkkoarkkitehtuureista....3
2.2. Verkkoprotokollatyypit....4
2.3. Verkkopaketti....7
2.4. Verkkoprotokollien ominaisuuksista....9
2.5. Verkkoprotokollien teknisestä toteutuksesta....11
2.5.1. Bittitason operaatiot C-kielessä....11
2.5.2. Kokonaislukutyyppien binääriesityksestä....13
2.5.3. Tavujärjestyksistä....14
3. Reaaliaikaisten internet-pelien verkkoprotokollien erityispiirteitä....16
3.1 Lähetettävän tiedon tarpeellisuuden arviointi....16
3.2. Verkkopakettien pakkaus....18
3.2.1. Ulkoiset pakkaustekniikat....18
3.2.2. Sisäiset pakkaustekniikat....19
3.3. Verkkoviiveen torjunta....24
3.4. Kokonaiskuva vaatimuksista ja ratkaisuista....28
4. Quake-sarjan historiaa....29
5. Ioquake3-protokollan keskeiset ominaisuudet....33
5.1. Käytetyt verkkotekniset ratkaisut....33
5.2. Verkkopaketin rakenne....34
5.3. Verkkokanavatietueen rakenne....37
5.4. Verkkoliikennöinti....38
5.4.1. Verkkokanavan avaus....41
5.4.2. Verkkokanavan funktioista....46
5.4.3. Luotettavuuden hallinta....47
5.4.4. Verkkopakettien sirpalointi....49
5.5. Verkkoliikenteen minimointi....52
5.5.1. Kiinnostuksenhallinta....52
5.5.2. Delta-pakkaus....53
5.5.3. Huffmanin koodaus....59
5.6. Verkkoviiveen kompensointimenetelmät....63
6. Kehitysehdotuksia....65
6.1. Merkkijonomuotoiset viestit....65
6.2. Mukautuva Huffmanin koodaus....66
6.3. Lähdekoodin luettavuus....67
7. Loppusanat....68
Viiteluettelo....70

Liitteet

Liite 1: Ioquake3:n staattinen Huffmanin puu
Liite 2: Ioquake3:n lähdekoodimuutokset

1. Johdanto

Reaaliaikaiset internet-pelit ovat koko muutaman vuosikymmenen olemassaolonsa ajan kasvattaneet suosiotaan, eikä kehityssuunnalle Suomessakaan näy toistaiseksi loppua. Pelialan nauttiessa poikkeuksellisen suurta huomiota lienee perusteltua olettaa monien kokeilunhaluisten ohjelmoijanalkujen olevan kiinnostuneita pelaamisen lisäksi myös pelien kehittämisestä. Pelaamisen hauskuudesta huolimatta tietokone- ja konsolipelit ovat pohjimmiltaan varsin monimutkaisia ja vaikeaselkoisia ohjelmia, joiden kehittäminen vaatii sekä kiinnostusta että asiantuntemusta. Reaaliaikaisilla internet-peleillä on lukuisten muiden pelilajityyppien tapaan omat erityspiirteensä, jotka yleensä liittyvät verkko-ohjelmointiin.

Reaaliaikaiset internet-pelit asettavat pelinkehittäjälle joukon vaatimuksia, joiden täyttäminen edellyttää sekä tiettyjä pohjatietoja että sujuvaa ohjelmointitaitoa. Pelin luonne ja laji määräävät luonnollisesti osan toteutettavista vaatimuksista, mutta tietyt peruspiirteet on toteutettava lähes aina. Jotta reaaliaikaisille internet-peleille ominaisten piirteiden toteuttaminen olisi mahdollista, on oleellista tuntea myös pelin verkkoprotokollan pohjana toimivien alemman tason protokollien ominaisuudet ja toimintaperiaatteet. Reaaliaikaisen internet-pelin kehityksen voidaankin sanoa alkavan samoista asioista kuin minkä tahansa korkean tason internet-sovelluksen.

Tässä tutkielmassa tarkastellaan reaaliaikaisten internet-pelien verkkotekniikkaa aloittaen internetissä tapahtuvan tiedonsiirron perusteista ja päätyen lopulta yksityiskohtaiseen internet-pelin verkkotekniikan lähdekoodianalyysiin. Tutkielman tarkoituksena on selvittää reaaliaikaisten internet-pelien verkkoteknisiä erityispiirteitä ja niiden käytännön toteutusta. Lähdekoodianalyysin tavoitteena on dokumentoida internet-peleihin yleisesti sovellettavissa olevia kiinnostavia verkkoteknisiä ratkaisuja sekä näiden toteutukseen käytettäviä matalan tason ohjelmointitekniikoita.

Yleinen verkkoyhteyksien käsittely aloitetaan luvussa 2 muun muassa erilaisten verkkoarkkitehtuurien ja verkkoprotokollatyyppien sekä protokollien matalan tason ohjelmointitekniikoiden tarkastelulla. Tämän jälkeen luvussa 3 syvennytään reaaliaikaisten internet-pelien erityisvaatimuksiin. Luvussa 4 käsitellään hieman Id Softwaren suosittua ja reaaliaikaisten internet-pelien historian kannalta merkittävää Quake-sarjaa, minkä jälkeen luvussa 5 siirrytään lähdekoodianalyysiin. Lähdekoodianalyysin kohteena on verkkotekniseltä toteutukseltaan monipuolinen ioquake3 [The ioquake Group, 2014].

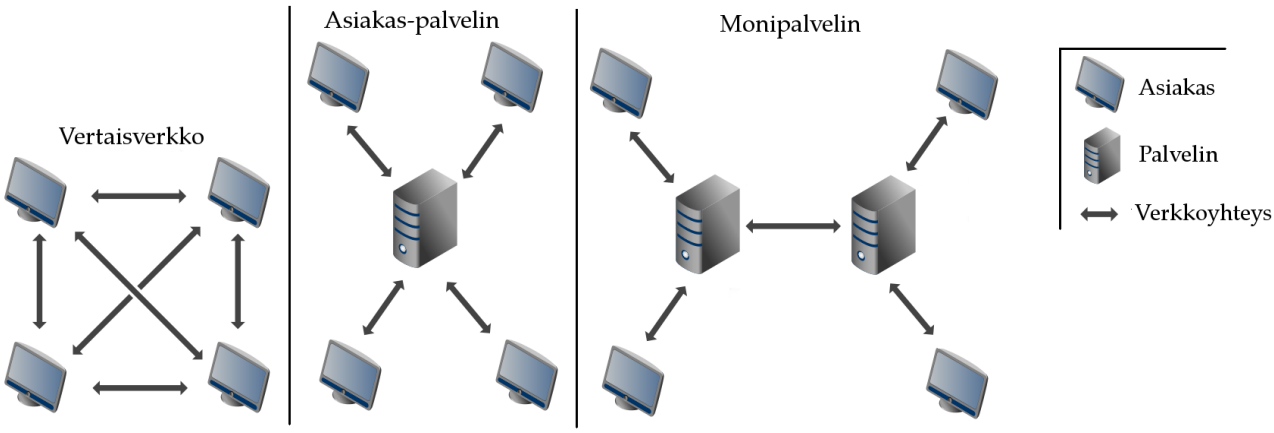
Luvussa 6 kootaan yhteen analyysin perusteella muodostettuja kehitysehdotuksia. Luku 7 sisältää lyhyen yhteenvedon tutkielman annista.

2. Verkkoyhteyksistä

Tietokoneiden välillä tapahtuvan kommunikaation on viestien ymmärrettävyyden varmistamiseksi noudatettava ennaltamääritettyä kommunikaatiotapaa, jota kutsutaan protokollaksi. Internet-liikenteessä tähän tarkoitukseen käytetään yleisimmin monitasoista TCP/IP-protokollaperhettä, joka mahdollistaa muun muassa eri kokoisten, eri merkkisten ja eri käyttöjärjestelmillä toimivien tietokoneiden välisen kommunikaation [Stevens, 1993]. Verkkoyhteyttä käyttävää ohjelmaa toteutettaessa täytyy verkkoprotokollan valinnan lisäksi tehdä myös korkeamman tason valintoja eri osapuolten tavasta kommunikoida keskenään [Stevens *et al.*, 2003]. Aloitetaan verkkoyhteyksien tarkastelu yleisellä tasolla perehtymällä ensin edellä mainittuihin korkean tason ratkaisuihin.

2.1. Verkkoarkkitehtuureista

Verkkoarkkitehtuurilla eli kommunikaatioarkkitehtuurilla tarkoitetaan korkean tason ratkaisua, jota noudattaen verkon eri laitteet kommunikoivat keskenään [Smed *et al.*, 2002a; 2002b]. Perinteisimmät arkkitehtuurit verkkopeleissä ovat tyypillisesti olleet vertaisverkko ja erityisesti asiakas-palvelin -malli [Bonham *et al.*, 2000]. Varsinkin nykyisissä massiivimoninpeleissä käyttöön on kuitenkin otettu myös hajautettu monipalvelin-arkkitehtuuri (eng. distributed multiserver architecture) [Yahyavi & Kemme, 2013], jossa palveluntarjoajan taakka hajautetaan usealle eri palvelimelle. Yahyavin ja Kemmen [2013] mukaan pelikehittäjät valitsevat kuitenkin useimmiten perinteisen asiakas-palvelin -mallin etenkin sen helpon ylläpidettävyyden ja hallinnan ansiosta. Valitulla verkkoarkkitehtuurilla on suuri merkitys sekä verkkoyhteyttä käyttävän sovelluksen kehitystyön että lopullisen käyttökokemuksen kannalta. Eri verkkoarkkitehtuureita on havainnollistettu kuvassa 1.



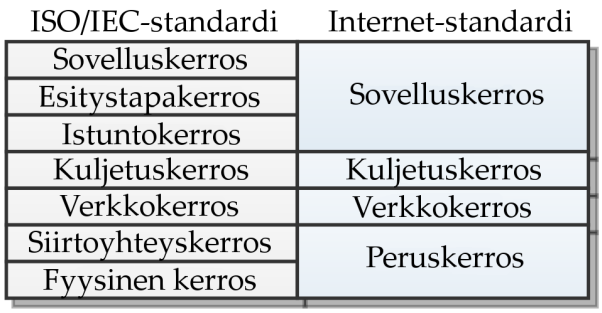
Kuva 1. Vertaisverkko-, asiakas-palvelin- ja monipalvelinverkkoarkkitehtuurit.

Monipalvelinarkkitehtuuri voi Yahyavin ja Kemmen [2013] mukaan toimia kahdella eri periaatteella: palvelimet voivat toimia itsenäisesti, jolloin palvelimien pelimaailmat eivät ole yhteydessä toisiinsa, tai vaihtoehtoisesti jakaen saman pelimaailman osiin, jolloin kukin palvelin vastaa tietystä osasta samaa pelimaailmaa. Palvelimien toimiessa itsenäisesti voidaan kyseessä ajatella olevan perinteinen asiakas-palvelin -arkkitehtuuri sillä poikkeuksella, että toinen versio samasta pelimaailmasta on olemassa myös toisella palvelimella [Yahyavi & Kemme, 2013].

Ensimmäiset reaaliaikaiset verkkopelit – kuten Id Softwaren Doom [Armitage *et al.*, 2006; van Waveren, 2006] – toimivat vertaisverkkoperiaatteella, mikä osoittautui silloisella tekniikalla vääräksi ratkaisuksi internet-pelaamisen yleistyttyä [Bonham *et al.*, 2000]. Vertaisverkkomallin soveltuvuutta internet-peleihin on kuitenkin alettu tutkia uudelleen, sillä se on huomattavasti skaalautuvampi kuin asiakas-palvelin- tai jopa monipalvelinarkkitehtuuri [Yahyavi & Kemme, 2013]. Myös monipalvelinarkkitehtuurin käyttöä perinteisemmissä internet-peleissä kuten Id Softwaren Quake 2:ssa [Bharambe *et al.*, 2006] ja Quake 3:ssa [Ploss *et al.*, 2008] on tutkittu. Kuten Yahyavi ja Kemme [2013] toteavat, asiakas-palvelin -mallia voidaan kuitenkin yhä pitää keskeisimpänä verkkoarkkitehtuurina reaaliaikaisia internet-pelejä käsiteltäessä.

2.2. Verkkoprotokollatyypit

Käytettävän verkkoarkkitehtuurin lisäksi tärkeä elementti pelien verkkoteknisessä toteutuksessa ovat käytetyt verkkoprotokollat. Verkkoprotokollat voidaan luokitella eri tasoille noudattaen esimerkiksi ISO/IEC-standardissa 7498-1 määriteltyä Open Systems Interconnection (OSI) -referenssimallia [ISO, 1994] tai internet-standardin [Braden, 1989] mukaista mallia. OSI-mallin seitsemänkerroksinen jako voidaan suhteuttaa internet-standardissa määriteltyyn neljäkerroksiseen jakoon kuvassa 2 esitetyllä tavalla [Braden, 1989; Stevens *et al.*, 2003].



Kuva 2. ISO/IEC-standardin OSI-mallin ja internet-standardin mukaiset verkkoprotokollapinot [ISO, 1994; Braden, 1989; Stevens *et al.*, 2003].

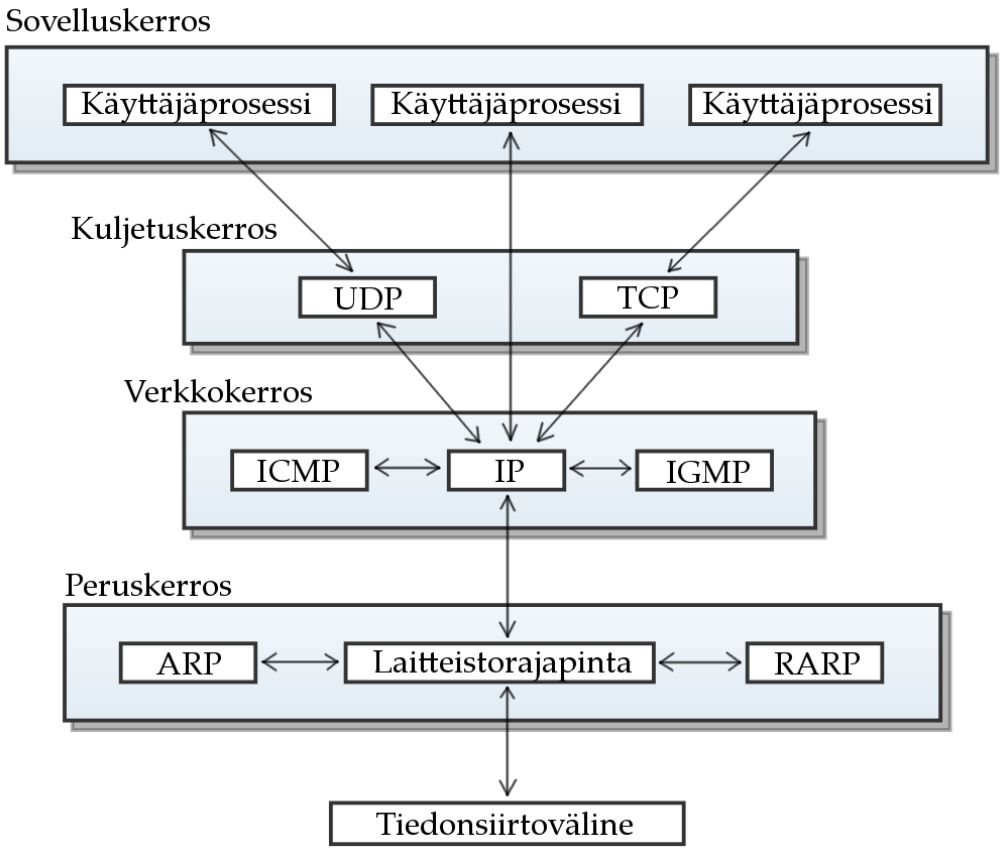
OSI-mallin [ISO, 1994] ja internet-standardin [Braden, 1989] määritelmissä annetaan tiukat rajoitteet kuvassa 2 esitettyjen protokollapinojen kerroksille, mutta niiden läpikäyminen yksityiskohtaisesti ei ole reaaliaikaisten internet-pelien tutkimuksen kannalta olennaista. Lisäksi alan kirjallisuudessa näyttää olevan epäselvyyksiä OSI-mallin asettamista rajoista kuhunkin kerrokseen kuuluvien protokollien suhteen: esimerkiksi Li ja muut [2011] mainitsevat esitystapakerroksen käsittävän muun muassa tiedon salaus- ja pakkausmekanismeja, mutta OSI-mallin kehitykseen syvällisesti perehtyneen Dayn [1996] mukaan tämä on vain yksi OSI-mallin kerroksiin liittyvistä yleisistä harhakäsityksistä.

Kaiken kaikkiaan reaaliaikaisten internet-pelien aihealueessa on loogisempaa keskittyä internet-standardin määrittelemään protokollapinoon, sillä TCP/IP-protokollaperhe noudattaa tarkalleen internet-standardin määrittelemää mallia [Stevens, 1993]. Tämä on oleellista, koska internet-pelien verkkoprotokollat käyttävät sovelluskerroksen alapuolella tiedonsiirtoon tyypillisesti TCP/IP-protokollaperheen protokollia [Yahyavi & Kemme, 2013]. TCP/IP-protokollaperheen ja OSI-protokollien välistä vastakkainasettelua ovat käsitelleet muun muassa Maathuis ja Smit [2003].

Reaaliaikaisten internet-pelien kannalta internet-standardin protokollapinon kerroksista erityisen huomionarvoinen on sovelluskerros, johon muun muassa pelien omat verkkoprotokollat kuuluvat. Muita tähän kerrokseen kuuluvia tavanomaisia protokollia ovat muun muassa FTP (File Transfer Protocol) ja telnet [Braden, 1989]. Myös kuljetuskerroksen protokollat ovat aihealueen kannalta oleellisia, sillä kuten edellä mainittiin, sovelluskerroksen protokollat käyttävät perustoiminnallisuudessaan apuna alempien kerroksien protokollia. Oleellista onkin ymmärtää, että verkkoprotokollapinoissa ylempänä olevien protokollien toiminnallisuus rakentuu aina alemman kerroksen protokollan tarjoamalle pohjalle [ISO, 1994; Braden, 1989].

Internet-standardin protokollapinon verkkokerroksen (tunnetaan myös kuvaavasti internet-kerroksena [Braden, 1989]) ja varsinkin peruskerroksen protokollat ovat niin matalan tason kommunikaatiokanavia, että niiden toimintaperiaatteiden syvällinen ymmärtäminen ei ole internet-pelin verkkoprotokollan ohjelmoinnin kannalta kovinkaan olennaista. Tämä johtuu siitä, että protokollien välinen hierarkia toimii automaattisesti, eikä alempien kerrosten protokollien toimintaa tarvitse ohjata erikseen. On kuitenkin tarpeen olla tietoinen alempien kerrosten protokollien ominaisuuksista ja rajoituksista, jotka saattavat vaikuttaa sovelluskerroksen protokollien toimintaan [Stevens *et al.*, 2003].

Kuva 3 havainnollistaa internet-standardin protokollapinon eri kerroksiin kuuluvia TCP/IP-protokollaperheen protokollia.

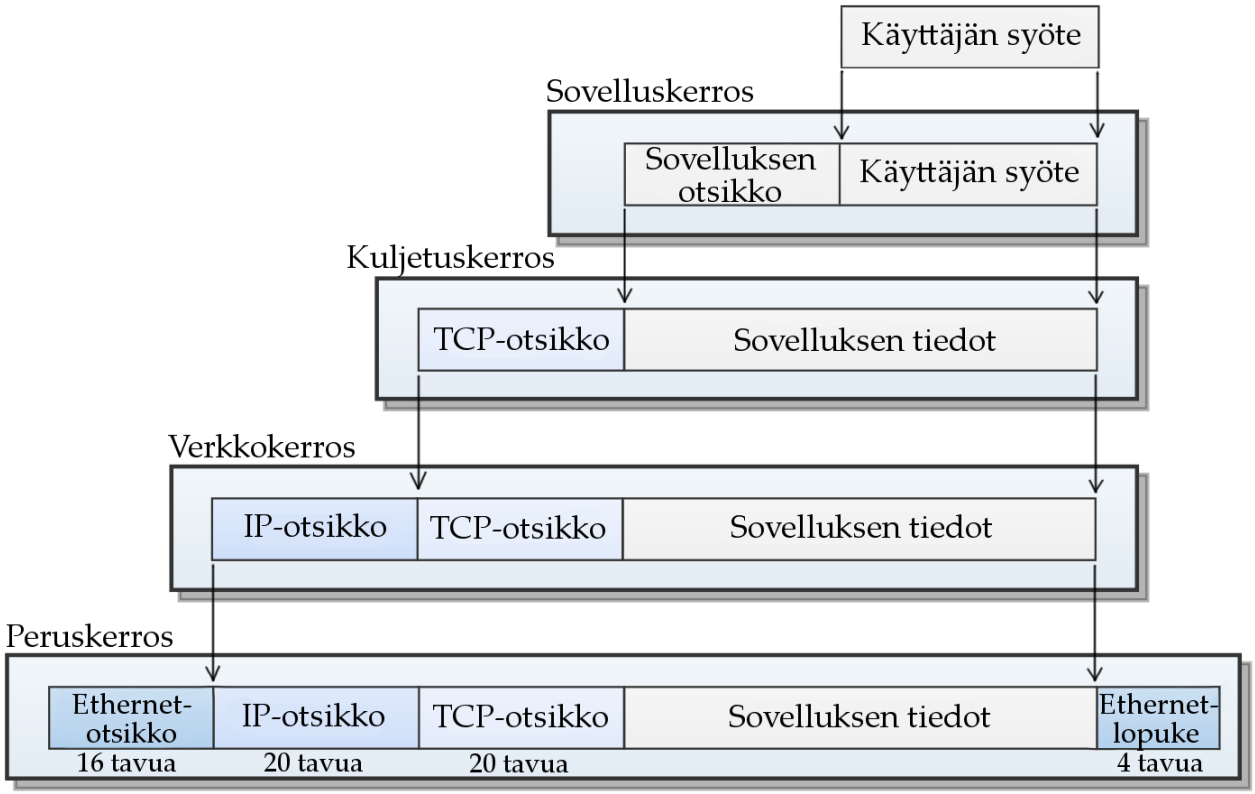


Kuva 3. Erilaisia verkkoprotokollia TCP/IP-protokollaperheen eri kerroksissa [Stevens, 1993].

Kuvassa 3 esiintyvistä verkkoprotokollista olennaisia ovat varsinkin UDP (User Datagram Protocol), TCP (Transmission Control Protocol) ja IP (Internet Protocol). Reaaliaikaisten internet-pelien sovelluskohtaiset verkkoprotokollat pohjautuvat tyypillisesti UDP- tai TCP-protokollaan [Harcsik *et al.*, 2007], jotka kummatkin pohjautuvat vuorostaan IP-protokollaan. IP-protokollaa käytetään viestien kuljetukseen tietokoneiden välillä, kun taas UDP tai TCP vastaavat viestien toimittamisesta oikeaan määränpäähän kohdetietokoneessa [Comer, 2000]. Huomionarvoista on, että IP-protokollasta on olemassa kaksi nykyään käytössä olevaa versiota: IPv4 ja vielä yleistymässä oleva IPv6 [Stevens *et al.*, 2003]. Mikäli toisin ei erikseen mainita, tässä tutkielmassa IP-protokollalla viitataan nimenomaan vanhempaan IPv4-versioon. Verkkokerroksen alla oleva peruskerros koostuu tyypillisesti tietokoneen verkkolaitteistosta ja sen ajureista [Stevens *et al.*, 2003].

2.3. Verkkopaketti

Verkkoprotokollien käsittelemiä tietoyksiköitä kutsutaan yleisesti verkkopaketeiksi, joskin muun muassa UDP- ja IP-protokollien kohdalla käytetään myös käsitettä datagrammi (eng. datagram) [Stevens, 1993; Comer, 2000]. Stevensin [1993] mukaan datagrammi viittaa nimenomaan epäluotettavan protokollan (ks. kohta 2.4.) käsittelemiin tietoyksiköihin, mutta johdonmukaisuuden nimissä lienee tässä perusteltua käyttää kaikille protokollille yhteistä nimitystä verkkopaketti. Verkkopaketti muodostuu, kun käyttäjän sovellukselle antama syöte aloittaa tapahtumaketjun, jossa jokaisen kerroksen protokolla lisää verkkopakettiin oman toimenkuvansa mukaiset tiedot. Tätä prosessia on havainnollistettu kuvassa 4.



Kuva 4. Verkkopaketin muodostuminen käyttäjän syötteestä protokollapinon alimpaan kerrokseen [Stevens, 1993]. Esimerkissä on käytetty kuljetuskerroksessa TCP-protokollaa.

Kuten kuvasta 4 nähdään, verkkopaketin koko kasvaa, kun jokaisen protokollapinon kerroksen protokolla lisää pakettiin oman osuutensa tarvittavasta informaatiosta. Sovelluskerroksessa käyttäjän syöte muunnetaan oletettavasti ohjelman ymmärtämäksi komennoksi ja se lähetetään eteenpäin sovelluksen oman verkkoprotokollan otsikon kera. Kuljetuskerroksessa käytetään TCP-protokollaa, joka lisää verkkopakettiin oman otsikkonsa. TCP-otsikko sisältää muun muassa porttinumeron, jolla verkkoyhteyttä

käyttävä sovellus yksilöidään [Stevens, 1993]. Verkkokerroksen IP-protokolla lisää niin ikään oman otsikkonsa, joka sisältää muun muassa lähettävän ja vastaanottavan tietokoneen IP-osoitteet. Peruskerroksessa lisätään viimeiseksi ethernet-protokollan tarvitsemat tiedot kuten muun muassa lähettävän ja vastaanottavan tietokoneen fyysiset eli niin sanotut MAC-osoitteet [Stevens, 1993]. On oleellista ymmärtää, että verkkopaketti kokonaisuutena koostuu selvästi erillisistä osista, joita luetaan ja kirjoitetaan eri protokollien avulla.

Reaaliaikaisten internet-pelien tapauksessa kuvan 4 esimerkin käyttäjän syöte voisi olla esimerkiksi näppäimen painallus, joka sovelluskerroksessa muunnetaan komennoksi liikuttaa pelihahmoa. Kun pelin oma verkkoprotokolla on lisännyt oman otsikkonsa, verkkopaketti lähtee protokollapinoa pitkin alaspäin kunnes se peruskerroksen jälkeen lähetetään eteenpäin esimerkiksi verkkokaapelia pitkin. Verkkopaketti matkaa vastaanottajalle pitkin internet-reitittimiä, jotka käsittelevät verkkopaketin tiedot peruskerroksesta verkkokerroksen IP-protokollaan asti [Stevens, 1993]. Koska verkkokerroksen IP-protokolla tarjoaa tarvittavat tiedot verkkopaketin edelleenlähetykseen [Stevens, 1993], korkeamman kerroksen protokollia ei reitityksen tarvita. Vastaanottajan päässä verkkopaketti puretaan ja tulkitaan peruskerroksesta sovelluskerrokseen, jotta verkkosovellus voi suorittaa halutut komennot. Tässä tapauksessa kyseinen verkkosovellus on internet-pelin asiakas- tai palvelinohjelma.

Olennaisia verkkoprotokollien määrittämiä rajoituksista ovat muun muassa verkkopaketin minimikoko sekä maksimikoko, josta käytetään myös termiä MTU (eng. maximum transmission unit). Esimerkiksi IPv4-protokollalla verkkopaketin maksimikoko (MTU) mukaan lukien otsikko on 65535 tavua ja minimikoko 68 tavua [Stevens *et al.*, 2003]. Peruskerroksen ethernet-protokollan määrittelemä verkkopaketin maksimikoko sen sijaan on vain 1500 tavua [Stevens *et al.*, 2003], mikä on oleellinen tieto korkeamman tason protokollien suunnittelussa. Ethernet-protokollan asettaman rajoitteen kiertämiseksi IP-protokolla jakaa IP-verkkopaketin sirpaleiksi (eng. fragment), jotka kootaan tavallisesti takaisin yhteen vasta lopullisessa määränpäässä [Stevens *et al.*, 2003].

Eri verkkoprotokollat vaativat verkkopaketista eri määrän tilaa tarvittavan informaation tallentamiseen. Koska muiden kuin sovelluskerroksen protokollien lisäämä tieto on lähinnä verkkopaketin perille saattamisen kannalta välttämätöntä informaatiota varsinaisen viestin sijaan, voidaan eri otsikoista muodostuvaa verkkopaketin osaa luonnehtia sanalla yleiskustannus (eng. overhead) [Stevens, 1993]. Esimerkiksi pelihahmon liikuttamiseen

tarkoitetun komennon lähettäminen vaatisi kuvan 4 protokollapinossa yhteensä 60 tavun yleiskustannuksen. Kuvan 4 esimerkissä sovelluskerroksen protokollan otsikon kokoa ei kuitenkaan ole määritelty, joten todellisuudessa yleiskustannus olisi esimerkin verkkoprotokollilla suurempi. Toki on myös mahdollista tai jopa tavanomaista, että sovelluksella ei ole varsinaista omaa protokollaa, vaan se käyttää suoraan esimerkiksi TCP-protokollaa.

2.4. Verkkoprotokollien ominaisuuksista

Verkkoprotokollapinon eri kerrosten protokollat lisäävät verkkopaketteihin erilaista tietoa, mikä johtuu ensisijaisesti protokollien erilaisista toimenkuvista. On kuitenkin syytä huomioida, että myös samassa verkkoprotokollapinon kerroksessa sijaitsevat protokollat voivat erota ominaisuuksiltaan huomattavasti, vaikka korkealla tasolla toimenkuva olisi sama. Esimerkiksi kuljetuskerroksen TCP- ja UDP-protokollat eroavat toisistaan oleellisesti, sillä toinen niistä on niin sanotusti luotettava ja toinen epäluotettava protokolla. Luotettavuuden käsitettä onkin syytä tarkastella lähemmin.

Matalimmalla tasolla tietokoneverkot ovat epäluotettavia – tietoa voi kadota tai tuhoutua, ja lähetetyt paketit voivat saapua vastaanottajalle sattumanvaraisessa järjestyksessä [Comer, 2000]. Edellä mainitut piirteet voidaan tiivistää sanomalla, että tietokoneverkossa käytetty verkkoprotokolla on yhteydetön ja epäluotettava.

Yhteydettömyys viittaa protokollan kyvyttömyyteen säilyttää tilatietoja peräkkäin saapuvista paketeista. Toisin sanoen kaikki paketit käsitellään itsenäisesti toisista paketeista riippumatta, minkä ansiosta alun perin peräkkäin lähetetyt paketit saattavat saapua eri järjestyksessä eri reittejä pitkin [Stevens, 1993]. Yhteydettömyyden käsitteeseen liittyy myös niin kutsuttu kaistan ulkopuolinen liikenne (eng. out-of-band tai expedited). Tällaisella liikenteellä tarkoitetaan kiireellistä verkkoliikennettä, joka on toimitettava perille ennen kaikkea normaalisti lähetettävää liikennettä [Stevens *et al.*, 2003]. Käytännössä kaistan ulkopuoliset paketit lähetetään yhteydettömän tiedonsiirron toimintaperiaatteen mukaisesti kaikki lähetysjärjestyskäytännöt sivuuttaen. Yhteydellinenkin protokolla voi siis toimia yhteydettömässä tilassa lähettämällä tai vastaanottamalla kaistan ulkopuolista liikennettä.

Protokollan epäluotettavuudella taas tarkoitetaan, että verkkoprotokolla ei voi taata pakettien saapumista perille. Korkean tason sovellukset vaativat yleensä luotettavaa tiedonsiirtoyhteyttä, minkä toteuttaminen verkkoprotokollapinon sovelluskerroksessa on

Comerin [2000] mukaan ohjelmoijalle vaativaa ja raskasta. Internet-standardin [Braden, 1989] protokollapinon kuljetuskerroksen protokollista esimerkiksi TCP-protokollaan onkin lisätty toiminnallisuutta, joka takaa verkkoyhteyden luotettavuuden. Toisaalta taas samaan kerrokseen kuuluva UDP-protokolla on epäluotettava ja huomattavasti yksinkertaisempi. UDP:n ja TCP:n kuljetuskerroksen alapuolella olevan verkkokerroksen IP-protokolla on luonnollisestikin epäluotettava [Stevens, 1993].

Vaikka TCP-protokolla tarjoaakin luotettavan ja nykyään laajalti käytetyn tiedonsiirtoyhteyden, kaikkiin käyttötarkoituksiin se ei kuitenkaan sovellu. Joissakin tapauksissa tiedonsiirtoyhteys täytyy Stevensin [1993] luonnehtimalla tavalla rakentaa epäluotettavan protokollan päälle lisäten luotettavuus ja muu haluttu lisätoiminnallisuus sovelluskerroksessa. Reaaliaikaiset internet-pelit ovat yksi esimerkki tällaisista sovelluksista.

TCP-protokollan luotettavuus perustuu kadotetun tiedon uudelleenlähettämiseen, kunnes vastaanottaja vahvistaa tiedon saapuneen [Comer, 2000]. Tämä ei ole toivottava toimintatapa reaaliaikaisten internet-pelien tapauksessa. Kuten muun muassa van Waveren [2006] sekä Bonham ja muut [2000] toteavat, Quaken kaltaisissa peleissä matkalla kadonnut tai väärässä järjestyksessä saapuva tieto on käytännössä käyttökelvotonta, sillä mahdollisen uudelleenlähetyksen jälkeen tieto on joka tapauksessa jo täysin vanhentunutta. Tämän johdosta internet-pelit käyttävät yleensä verkkoprotokollapinon kuljetuskerroksessa yksinkertaisempaa UDP-protokollaa. Tarvittava lisätoiminnallisuus ohjelmoidaan pelin verkkokoodiin sovelluskerroksessa, jolloin pelin oma verkkoprotokolla huolehtii sovellukselle ominaisten vaatimusten täyttymisestä.

Verkkoprotokollan luotettavuuden ja yhteydellisyyden lisäksi korkeamman tason protokollat kuten TCP tarjoavat muutamia muita olennaisia ominaisuuksia. Tällaisia ovat muun muassa vuonohjaus (eng. flow control), ruuhkanhallinta (eng. congestion control) ja verkkopakettien vastaanoton kuittaus (eng. acknowledgment/ack).

Vuonohjauksella viitataan tekniikkaan, jolla varmistetaan tiedonlähetyksen optimaalinen nopeus ottaen huomioon sekä lähettävän että vastaanottavan laitteen resurssit [Stevens, 1993]. Toisin sanoen tekniikalla estetään esimerkiksi erittäin nopeaa tietokonetta lähettämään verkkopaketteja nopeammin kuin käytössä oleva verkko pystyy niitä käsittelemään [Comer, 2000].

Ruuhkanhallinnalla taas viitataan tekniikkaan, jolla jo tapahtunut liikenteen ruuhkaantuminen hallitaan. Esimerkiksi TCP-protokolla tunnistaa yhteyden ruuhkaantuneen tyypillisesti verkkoyhteyden aikakatkaisun tai saman kuittauksen toistumisen perusteella [Stevens, 1993]. Aikakatkaisulla viitataan tilanteeseen, jossa lähetettyä verkkopakettia ei kuitata vastaanotetuksi tietyn aikamääreen puitteissa [Stevens, 1993].

Sovelluskerroksen protokollissa korkeamman tason toiminnallisuus voi toisinaan olla erotettu selvästi erilliseksi kokonaisuudeksi, joka muodostuu tyypillisesti kokoelmasta erilaisia tietueita, luokkia ja funktioita. Reaaliaikaisista internet-peleistä esimerkiksi Quake-sarjan pelit [Id Software, 1999; 2001; 2005] käyttävät yhteydellisen tilan tiedonsiirtoyhteydestä nimitystä verkkokanava (eng. net channel tai lyhyesti netchan). Tällaisesta kokonaisuudesta erotetut verkkoprotokollan funktiot ovat tyypillisesti toiminnallisuudeltaan lähellä protokollapinon alemman kerroksen protokollaa, eivätkä ne toteuta sovelluskerroksen protokollan korkeamman tason toiminnallisuutta.

2.5. Verkkoprotokollien teknisestä toteutuksesta

Kuten verkkoprotokollien toteutusta käsittelevää kirjallisuutta [Stevens, 1993; Stevens *et al.*, 2003] tarkasteltaessa nopeasti selviää, verkkoprotokollien käytännön ohjelmointityö koostuu usein bittitason ohjelmoinnista. Voitaneen olettaa, että syy tähän juontuu yhdestä verkko-ohjelmoinnin keskeisistä teemoista eli lähetettävän tiedon määrän minimoimisesta. Myös reaaliaikaisten internet-pelien verkkoprotokollien ohjelmoinnissa turvaudutaan usein bittitason ohjelmointiin, joten aihealueen selventäminen oleellisilta osin ohjelmointiteknisestä näkökulmasta lienee paikallaan.

Alakohta 2.5.1. toimikoon ohjenuorana verkkoprotokollien lähdekoodissa paljon käytettyjen bittitason operaatioiden tulkinnassa. On myös hyödyllistä tarkastella hieman nykyaikaisten tietokoneiden tapaa käsitellä etumerkillisiä ja etumerkittömiä kokonaislukutyyppejä bittitasolla – tästä lisää alakohdassa 2.5.2. Bittitason ohjelmoinnin tarkastelussa keskitytään aihealueeseen C-ohjelmointikielen rajoissa. Alakohdassa 2.5.3. käsitellään tietokoneen tavujärjestyksen merkitystä verkkoprotokollien ohjelmoinnissa.

2.5.1. Bittitason operaatiot C-kielessä

Tehokkaan tiedonsiirron mahdollistamiseksi verkkoprotokollien ohjelmointiteknisessä toteutuksessa on monesti käytetty bittitason operaatioita, jotka mahdollistavat bittitason ohjelmoinnin C-kielellä. Bittitason operaatioita voidaan lähetettävien verkkopakettien koon

optimoinnissa pitää välttämättömänä työkaluna, sillä tarvittavan tiedon toimittamiseen riittää monesti vain muutama bitti esimerkiksi yhden tavun sijaan.

Taulukossa 1 on esitetty C-kielen käsittämät bittioperaattorit. Bittioperaattoreita voidaan soveltaa ainoastaan etumerkittömiin tai etumerkillisiin kokonaislukutyyppeihin eli C-kielen tietotyyppeihin char, short, int ja long [Kernighan & Ritchie, 1988].

&	Bittitason JA (bitwise AND)
|	Bittitason TAI (bitwise inclusive OR)
^	Bittitason poissulkeva TAI (bitwise exclusive OR)
<<	Bittien siirto vasemmalle (left shift)
>>	Bittien siirto oikealle (right shift)
~	Bittitason 1:n komplementti, unaarinen (ones complement, unary)

Taulukko 1. C/C++-kielen bittioperaattorit [Kernighan & Ritchie, 1988].

Tarkastellaan bittioperaattoreita vielä käytännön esimerkkien kautta. Bittitason JA eli operaattori & asettaa bitille arvon 1 ainoastaan, kun molempien operandien bitin arvo on 1, minkä ansiosta sitä käytetään Kernighanin ja Ritchien [1988] mukaan usein nollaamaan jokin joukko bittejä. Esimerkiksi `10011 & 10110 = 10010`. Bittitason TAIta eli operaattoria | taas käytetään yleensä bittien päälle asettamiseen, sillä se antaa bitille arvon 1 kumman tahansa operandin bitin arvo ollessa 1. Esimerkiksi `10011 | 10110 = 10111`. Bittitason poissulkeva TAI eli ^ asettaa bitin päälle operandien bittien arvojen poiketessa toisistaan. Bitti asetetaan pois päältä, kun arvot ovat samat. Esimerkiksi `10011 ^ 10110 = 00101`.

Bittien siirto-operaattorit >> ja << siirtävät vasemman operandin bittejä oikealle tai vasemmalle *n* määrän paikkoja, jossa *n* on oikean operandin määrittämä posiitivinen luku. Bittien siirto-operaatioiden kohdistuessa etumerkittömiin lukuihin tyhjilleen jääneet paikat täytetään nollilla. Esimerkiksi `10011 >> 2 = 00100` ja `10011 << 2 = 01100`. Bittitason 1:n komplementti eli operaattori ~ antaa kokonaisluvun komplementin eli asettaa jokaisen poissa päältä olevan bitin päälle ja päinvastoin. Esimerkiksi `~10011 = 01100`.

Matalan tason lähdekoodissa saattaa soveltuvilta osin esiintyä bittitason operaattoreita normaalien laskuoperaattorien sijasta. Tällä toimintatavalla pyritään maksimoimaan ohjelman tehokkuus, sillä joissakin tapauksissa laskuoperaatiolla ja bittitason operaatiolla saadaan toimintaperiaatteiden erosta huolimatta sama lopputulos. Tällainen tapaus on esimerkiksi jakojäännöksen laskeminen jaettaessa kahdeksalla.

Jakojäännöksen laskemiseen käytettävä laskuoperaattori on C-kielessä tavallisesti %, mutta tässä tapauksessa voidaan saman lopputuloksen saamiseksi käyttää myös bittitason operaattoria & ja toisena operandina kokonaislukua 7. Koska kokonaisluku seitsemää vastaava bittijono on 111 ja kahdeksalla jaollisissa kokonaisluvuissa ainoastaan merkitsevin bitti on päällä, operaatiot x & 7 ja x % 8 tuottavat saman lopputuloksen. Koska kahdeksan bittiä vastaa yhtä tavua, jakojäännös jaettaessa kahdeksalla on verkkoprotokollien ohjelmoinnissa monesti hyödyllinen. Operaation lopputuloksesta voidaan päätellä, onko jokin bittijono jaettavissa suoraan tavuihin vai ei.

On syytä huomioida, että nykyään ainakin GNU:n gcc-kääntäjä [GNU, 2014a] osaa ohjelmaa kääntäessä muuntaa tavalliset laskuoperaatiot tehokkaimpaan mahdolliseen muotoon automaattisesti. Tämä tapahtuu jopa käännettäessä ohjelmaa ilman erityisiä optimointiasetuksia. Gcc-kääntäjän tuottama lopputulos voi siis olla täysin vastaava riippumatta siitä, käytettiinkö bittitason operaatiota vai jakojäännösoperaatiota. [Sanglard, 2009]

2.5.2. Kokonaislukutyyppien binääriesityksestä

Etumerkilliset (eng. signed) ja etumerkittömät (eng. unsigned) kokonaislukutyypit ovat C/C++-kielessä bittisyydeltään samoja, minkä johdosta niiden määrittelyjoukot poikkeavat toisistaan. Esimerkiksi kahdeksanbittinen unsigned char saa määrittelyjoukon [0, 255], kun taas signed char määrittelyjoukon [-128, 127]. Tästä johtuen sama bittijono tuottaa eri kokonaisluvun riippuen siitä, onko käytetty kokonaislukutyyppi etumerkitön vai etumerkillinen. [Kernighan & Ritchie, 1988]

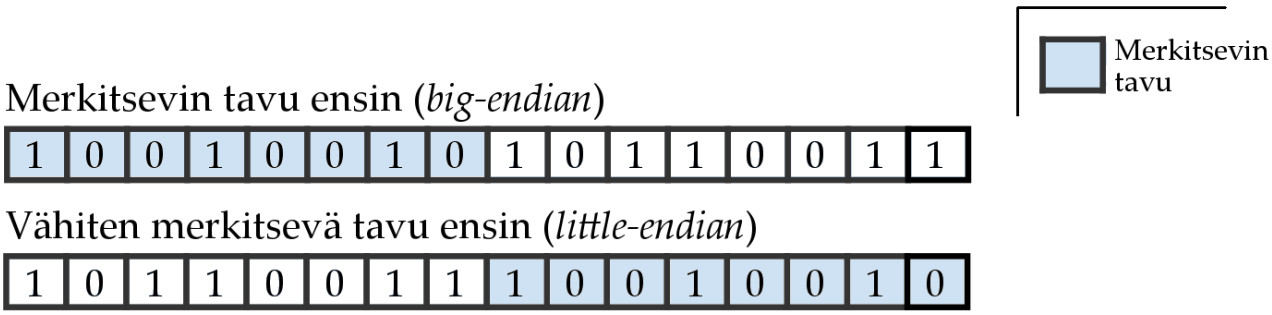
Etumerkillisten kokonaislukutyyppien binääriesitys on nykyisten kotitietokoneiden käyttämissä IA-32- ja AMD64-käskykannoissa määritelty kahden komplementti -periaatetta käyttäen [Intel, 2013; AMD, 2013]. Kokonaislukujen esitystavalla on verkkoprotokollien lähdekoodin ymmärtämisen kannalta ainakin yksi oleellinen sivuvaikutus: etumerkittömän kokonaislukutyypin maksimiarvo vastaa aina binääriesitykseltään saman etumerkillisen kokonaislukutyypin arvoa -1. Esimerkiksi tavu

11111111 vastaa unsigned char -kokonaislukutyypin maksimiarvoa 255, kun taas sama bittijono signed char -tyyppisenä tuottaa luvun -1 [Harris & Harris, 2012].

Luvun -1 sijoittaminen etumerkitöntä kokonaislukutyyppiä olevaan muuttujaan tuottaa saman bittijonon kuin etumerkillistä kokonaislukutyyppiä käytettäessä, vaikka muuttujan arvoa luettaessa saadut luvut eroavatkin toisistaan. Negatiivinen kokonaisluku muunnetaan C-kielessä etumerkittömään kokonaislukutyyppiin lisäämällä tai vähentämällä sijoitettavasta luvusta toistuvasti etumerkittömän tietotyypin maksimiarvo + 1, kunnes saatu arvo kuuluu tietotyypin määrittelyjoukkoon [ISO, 2011]. Näin ollen esimerkiksi luvun -1 sijoittaminen unsigned char -tyyppiseen muuttujaan tuottaa arvon 255, sillä -1 (sijoitettava luku) + 256 (tietotyypin maksimiarvo + 1) = 255.

2.5.3. Tavujärjestyksistä

Käsite tavujärjestys viittaa tietokoneen suorittimen tapaan käsitellä pidempiä kuin yhden tavun mittaisia kokonaisuuksia. Yleisesti käytössä olevia tavujärjestyskäytäntöjä on kaksi: eniten merkitsevä tavu ensin (eng. big-endian) ja vähiten merkitsevä tavu ensin (eng. little-endian) [Cohen, 1981]. Eri tavujärjestyksiä on havainnollistettu kuvassa 5.



Kuva 5. Desimaalijärjestelmän lukua 37555 vastaavat bittijonot eri tavujärjestyksiä käyttäen.

Koska bittijonon lukeminen väärää tavujärjestystä käyttäen voi muuttaa bittijonon tuottamaa arvoa, tavujärjestyksen huomioiminen verkkoprotokollien ohjelmoinnissa on ensiarvoisen tärkeää [Comer, 2000]. Mikäli verkkopaketin lähettäjän ja vastaanottajan tietokoneiden tavujärjestyskäytännöt eroavat toisistaan, verkkosovellus ei ilman tavujärjestyksen huomioimista todennäköisesti toimi oikein. Tästä johtuen alkuperäisessä internet-standardissa [Reynolds & Postel, 1994] kokonaislukujen lähetyksessä käytettäväksi tavujärjestykseksi määritellään eniten merkitsevän tavun lähettäminen ensin.

Internet-standardissa määritelty tavujärjestys juontanee juurensa standardin muodostamisen aikaisiin tietokoneisiin, jotka käyttivät pääasiassa edellä mainittua (big-endian) tavujärjestystä [Comer, 2000]. Nykyisin yleisessä käytössä olevat IA-32-, IA-64- ja AMD64-suoritinarkkitehtuurit käyttävät kuitenkin päinvastaista tavujärjestystä, jossa vähiten merkitsevä tavu tulee ensin [AMD, 2013; Intel, 2013]. Comerin [2000] mukaan standardissa määriteltävää tavujärjestystä huomattavasti tärkeämpää on kuitenkin itse standardin olemassaolo. Tähän päätelmään on helppo yhtyä, sillä oleellista verkkoprotokollien ohjelmoinnissa onkin lähinnä varmistaa, että bittijonoa lukiessa tavujärjestys vastaa aina paikallisen suoritinarkkitehtuurin tavujärjestystä. Paras keino tähän lienee juuri verkkopakettien lähettäminen tiettyä tavujärjestystä noudattaen riippumatta lähettäjän tai vastaanottajan tietokoneen paikallisesta tavujärjestyksestä.

Mikäli toisin ei erikseen mainita, tässä tutkielmassa esiintyvät bittijonot noudattavat big-endian -tavujärjestystä.

3. Reaaliaikaisten internet-pelien verkkoprotokollien erityispiirteitä

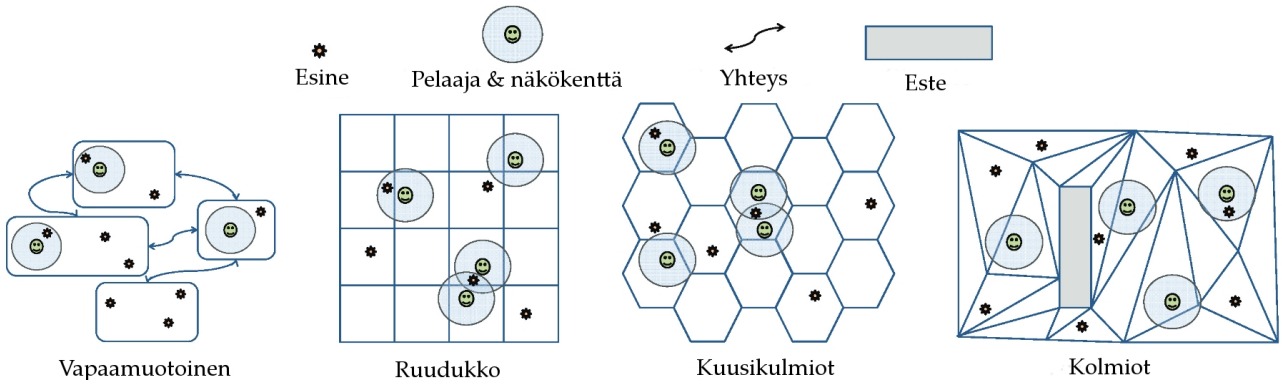
Seuraavaksi käsitellään reaaliaikaisten internet-pelien verkkoprotokollien kannalta oleellisia haasteita ja niiden ratkaisuja yleisellä tasolla. Huomionarvoista on, että useaa tässä luvussa esiteltyä tekniikkaa voisi olla perusteltua käyttää mitä vain verkkoprotokollaa toteutettaessa. Reaaliaikaisten internet-pelien tapauksessa poikkeuksellisen tehokkaat verkkotekniset ratkaisut ovat kuitenkin välttämättömyys – näin ei kaikkien verkkosovellusten kohdalla välttämättä ole.

Reaaliaikaiset internet-pelit asettavat käytettävälle verkkoprotokollalle tiettyjä erityisvaatimuksia, joita täyttämään on kehitetty monenlaisia lähestymistapoja [Smed *et al.*, 2002a; 2002b]. Aihealueeseen kuuluvat haasteet liittyvät perinteisesti lähinnä hitaiden verkkoyhteyksien sekä pelaajien ja palvelimen maantieteellisen etäisyyden aiheuttamien negatiivisten vaikutusten tasapainottamiseen [Pantel & Wolf, 2002; Huang & Griffioen, 2013]. On syytä huomioida, että vaikka hitaat verkkoyhteydet olivat ensimmäisten reaaliaikaisten internet-pelien aikaan merkittävä ongelma [Bonham *et al.*, 2000], niin nykyisten laajakaistayhteyksien ansiosta suurimman haasteen muodostanevat juuri maantieteelliset etäisyydet. Huangin ja Griffioenin [2013] mukaan peliyhtiöt valitsevatkin pelipalvelimien sijainnin tarkkaan harkiten.

Vaikka nykyiset laajakaistayhteydet mahdollistavat maantieteellisistä etäisyyksistä riippumatta suurehkojenkin verkkopakettien sujuvan lähettämisen ja vastaanottamisen, turhan tiedon lähettämistä on silti pidettävä negatiivisena asiana. Tarkastellaan seuraavaksi erilaisia verkkopakettien koon minimointiin kehitettyjä tekniikoita, jotka soveltuvat osaksi reaaliaikaisen internet-pelin verkkoprotokollaa.

3.1 Lähetettävän tiedon tarpeellisuuden arviointi

Ensimmäinen toimenpide lähetettävän tiedon minimoimiseksi lienee tyypillisesti tarpeettoman tiedon karsiminen. Lähetettävän tiedon tarpeellisuuden arviointi tehdään palvelimella yleensä asiakaskohtaisesti, sillä riippuen esimerkiksi pelaajan sijainnista pelikartalla eri pelaajat tarvitsevat erilaista tietoa pelin tilasta. Lähetettävän tiedon valikoimisesta käytetään yleisesti termiä kiinnostuksenhallinta (eng. interest management) [Yahyavi & Kemme, 2013]. Kiinnostuksenhallinnassa voidaan käyttää apuna erilaisia algoritmeja, jotka perustuvat pelimaailman kaavoittamiseen esimerkiksi joukoksi kolmioita, neliöitä tai kuusikulmioita [Boulanger *et al.*, 2006]. Erilaisia tapoja jakaa pelimaailma osiin on havainnollistettu kuvassa 6. Pelikohtaiset kiinnostuksenhallinnan kriteerit määrävät kutakin pelaajaa kiinnostavat sektorit.



Kuva 6. Erilaisia pelikartan kaavoitusmekanismeja [Yahyavi & Kemme, 2013].

Yleensä tietokonegrafiikkaan liittyvässä kirjallisuudessa esiintyvä termi PVS (eng. potentially visible set) viittaa tietystä näkökulmasta potentiaalisesti näkyvien polygonien joukkoon [Airey *et al.*, 1990], mutta tätä käsitettä sovelletaan myös verkko-ohjelmoinnissa [van Waveren, 2006; Abrash, 1997a]. Verkko-ohjelmoinnissa PVS-käsite voidaan venyttää tarkoittamaan esimerkiksi pelaajan näkökentässä olevien pelientiteettien joukkoa. Pelientiteeteillä tarkoitetaan tässä yhteydessä pelimaailmaan kuuluvia muuttuvia kohteita, joiden tilasta lähetetään tietoa asiakkaille verkon yli. Tyypillinen esimerkki tällaisesta entiteetistä on toinen pelaaja – tai tarkemmin määriteltynä toisen pelaajan pelihahmo eli avatar.

Esimerkiksi Quake-sarjan peleissä on perinteisesti käytetty pelikarttojen tallennukseen BSP-formaattia (eng. Binary Space Partitioning) [Abrash, 1997b], joka perustuu pelialueen jakamiseen kahtia, kunnes jako on suoritettu halutulla tarkkuudella [Abrash, 1997a]. Edellä mainitun prosessin tuloksena syntyy BSP-puu, jossa jokainen solmu vastaa jotakin pelikartan aluetta [Abrash, 1997a]. BSP-puita voidaan käyttää apuna potentiaalisesti näkyvien polygonien joukon eli PVS:n laskemiseen [van Waveren, 2007]. Grafiikan renderöintiä varten laskettuja BSP-solmukohtaisia PVS-arvoja voidaan käyttää hyödyksi myös verkkoteknisessä kiinnostuksenhallinnassa. Jos pelientiteetti on renderöintiin käytettävän PVS:n ulkopuolella, ei sitä myöskään yleensä tarvitse lähettää verkon yli [Abrash, 1997b].

Eräänlainen laajennus PVS-käsitteeseen on PHS (eng. potentially hearable set) [Abrash, 1997a], jonka voidaan verkko-ohjelmoinnissa käsittää tarkoittavan pelaajan sijainnista potentiaalisesti kuultavien pelientiteettien tai niiden aiheuttamien äänien joukkoa. Toisin sanoen joissakin tilanteissa voi olla tarpeen lähettää verkon yli tietoa entiteeteistä, jotka

eivät vielä näy pelaajan ruudulla, mutta ovat kuitenkin tarpeeksi lähellä vaikuttaakseen pelaajan tilanteeseen nyt tai lähitulevaisuudessa [Abrash, 1997a].

Kiinnostuksenhallinnan perusteella muodostettua tilanpäivityspakettia voidaan edelleen optimoida käyttämällä erilaisia verkkopaketin pakkaamiseen tarkoitettuja tekniikoita.

3.2. Verkkopakettien pakkaus

Verkkopakettien pakkaustekniikoilla saavutettavien tavoitteiden voidaan ajatella olevan samat pakkaustekniikasta riippumatta, mutta tekniikat voidaan tästä huolimatta jakaa toisistaan merkittävästi poikkeaviin ryhmiin. Smed ja muut [2002b] nimeävät kaksi eri ryhmää: sisäiset pakkaustekniikat ja ulkoiset pakkaustekniikat. Sisäisillä pakkaustekniikoilla viitataan tekniikoihin, jotka pakkaavat kunkin verkkopaketin riippumatta toisista verkkopaketeista [Smed *et al.*, 2002b]. Ulkoiset pakkaustekniikat taas saattavat käyttää hyödykseen esimerkiksi jo aikaisemmin lähetettyä informaatiota eli niillä on ulkoisia riippuvuuksia [Smed *et al.*, 2002b].

3.2.1. Ulkoiset pakkaustekniikat

Yksi esimerkki ulkoisesta pakkaustekniikasta on niin sanottu delta-pakkaus (eng. delta compression) [Bharambe *et al.*, 2008; van Waveren, 2006]. Delta-pakkauksella tarkoitetaan tekniikkaa, jolla voidaan optimoida palvelimen pelaajille lähettämien päivityspakettien koko. Tekniikan perusperiaate on ainoastaan muuttuneiden tietojen lähettäminen pelaajalle täyden pelitilanteen päivityksen sijasta.

Delta-pakkauksen kaltaisen tekniikan toteuttamiseksi palvelimella täytyy olla tieto siitä, mitä tietoa kullakin asiakkaalla on entuudestaan, sillä muuten muutosta suhteessa asiakkaan nykyiseen tilaan ei voida laskea. Tallennettua tietoa pelin tilasta (eng. game state) tiettynä hetkenä voidaan kutsua tilannevedokseksi (eng. snapshot) [Stefyn *et al.*, 2011; van Waveren, 2006]. Kun palvelin pitää kirjaa asiakkaan vastaanottamista ja kuittaamista verkkopaketeista, voidaan esimerkiksi liikkuvan kohteen absoluuttisten sijaintikoordinaattien sijasta lähettää vain muutos viimeisestä vahvistetusta tilannevedoksesta nykyiseen sijaintiin.

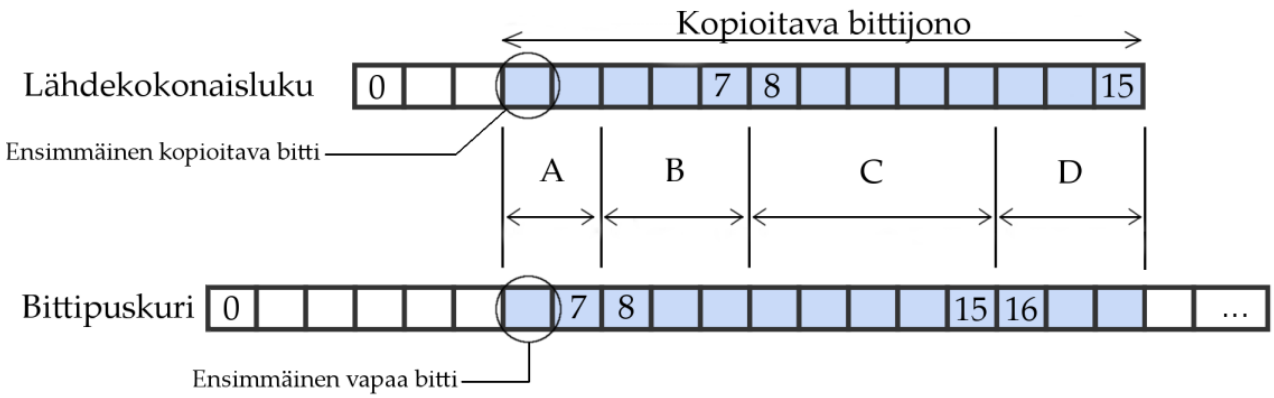
Koska muutoksen ilmoittamiseen tarvittavien lukujen voidaan olettaa olevan tyypillisesti pienemmät kuin absoluuttisen sijainnin tapauksessa [Smed *et al.*, 2002b], saadaan tällaisella tekniikalla pienennettyä verkkopakettien kokoa entisestään. On kuitenkin syytä huomioida, että mikäli lähetettäville luvuille on varattu kiinteä määrä tilaa, on

verkkopaketti syytä pakata vielä esimerkiksi jollakin bitinpakkaustekniikalla (ks. alakohta 3.2.2.). Delta-pakkaus ei siis yksinään välttämättä pienennä lähetettävien verkkopakettien kokoa.

3.2.2. Sisäiset pakkaustekniikat

Arvoltaan mahdollisimman pienien lukujen muodostamisen lisäksi yhtä oleellista on lukujen pakkaaminen mahdollisimman pieneen tilaan. Esimerkiksi kokonaisluvun 2 lähettäminen normaalina 16-bittisenä kokonaislukuna olisi verkkokapasiteetin haaskausta, koska saman luvun esittämiseen riittää 16 bitin sijaan kaksi bittiä: bittijono 10. Tiedon pakkaamiseksi mahdollisimman pieneen tilaan on kehitetty niin sanottuja bitinpakkaustekniikoita, jotka kuuluvat sisäisten pakkaustekniikoiden ryhmään. Oleellista bitinpakkaustekniikoissa verkkoprotokollien kohdalla on itse pakkaamisen lisäksi pakatun tiedon purkaminen etäkohteessa. Pakkauksen tapahtuessa eri tietokoneella kuin purkaminen on syytä ottaa huomioon esimerkiksi laitteistojen eroavaisuuksien aiheuttamat mahdolliset ongelmat.

Bitti kerrallaan esimerkiksi kokonaisluvun pakkaavan algoritmin toteutus on yksinkertaista, mutta tällaisen algoritmin suorituskykyä ei voida pitää riittävänä [Isensee, 2004]. Suorituskyvyn parantamiseksi algoritmin täytyykin tehdä pakkaus tavutasolla, mikä on huomattavasti monimutkaisempaa toteuttaa. Isensee [2004] esittelee pakkaus-algoritmin, jonka toteutuksessa on otettu huomioon tehokkuus, muistinkäyttö ja algoritmin sulautuvuus olemassaolevaan pelikoodiin. Tämän algoritmin toimintaa on havainnollistettu kuvassa 7.



Kuva 7. Bitinpakkausalgoritmi [Isensee, 2004].

Kuten alakohdassa 2.4.3. mainittiin, verkkoprotokollien lähettämä tieto on tavujärjestykseltään yleensä muotoa, jossa merkitsevin tavu tulee ensin. Tämän johdosta

myös Isenseen [2004] algoritmi muuntaa kopioitavan bittijonon big-endian -muotoon ennen kuvassa 7 esitetyn kopiointiprosessin alkamista. Etäkohteessa suoritettava purkurutiini muuntaa niin ikään bittijonon little-endian -muotoon, mikäli se on paikallisen tietokoneen suorittimen tavujärjestys. Näin bittijono on aina oikeassa muodossa paikalliselle tietokoneelle.

Kuvan 7 esimerkissä pakataan neljätavuinen kokonaisluku, jonka 16 bitistä 13 bittiä on merkitseviä ja näin ollen loput kolme bittiä turhaa tietoa. Isenseen [2004] pakkausalgoritmi jakaa kopioitavan bittijonon osiin, jotka määräytyvät joko lähteen tai kohteen tavurajan (eng. byte boundary) perusteella. Ennen bittijonon osiin jakamista bittijonot tasataan niin, että kopioitavan bittijonon ensimmäinen merkitsevä bitti ja bittipuskurin ensimmäinen vapaa bitti ovat linjassa. Kuvassa 7 kopioitava bittijono on jaettu neljään osaan, joista esimerkiksi ensimmäisen osan eli osan A raja määräytyy bittipuskurin tavurajan perusteella, kun taas osa B:n raja lähteen tavurajan perusteella.

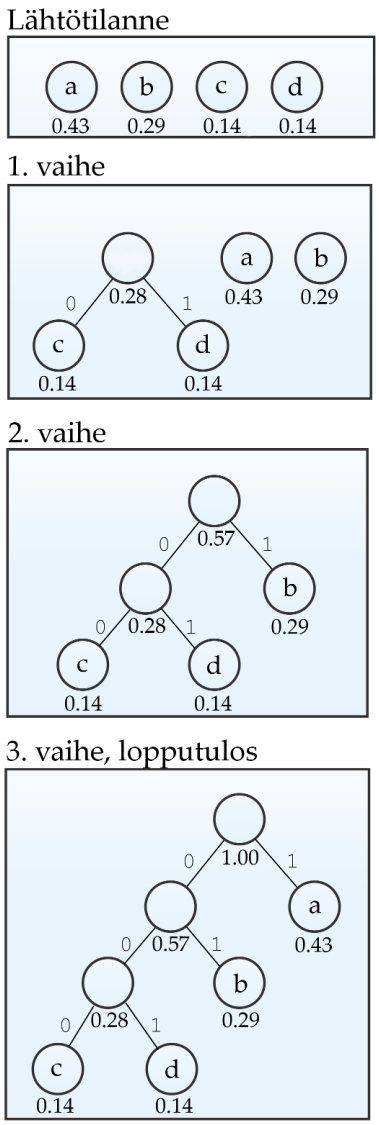
Algoritmi kopioi yhden osan kerrallaan, joten mikäli bittipuskurin ja lähteen tavurajat menevät yksi yhteen, voidaan kopiointi suorittaa jopa tavuittain [Isensee, 2004]. Pakatun bittijonon purkaminen tapahtuu saman periaatteen mukaisesti, joskin puskurin ja lähteen roolit ovat tällöin päinvastaiset. Algoritmin toimintaperiaatteen hahmotuttua voidaan huomata, että purkuprosessin tuloksena todella saadaan alkuperäistä lähdekokonaislukua arvoltaan vastaava bittijono.

On syytä huomioida, että eri verkkopeleissä käytettävät bitinpakkausalgoritmit ovat tyypillisesti suljettua lähdekoodia, eikä niiden toimintaperiaate välttämättä ole sama kuin edellä esitellyn Isenseen [2004] algoritmin. Hieman toisenlainen lähestysmistapa tiedon pakkaamiseen on Huffmanin koodaus, joka perustuu pakattavien kohteiden esittämiseen joukkona alkuperäistä vähemmän tilaa vievinä bittimuotoisina koodisanoina eli Huffmanin koodina [Sayood, 2012]. Esimerkiksi kahdeksanbittisistä merkeistä koostuvan merkkijonon yleisintä merkkiä vastaavaksi koodisanaksi voidaan määrittää 0, jolloin kyseinen merkki vie pakattuna vain yhden bitin yhden tavun sijaan. Tällainen lähestymistapa eroaa oleellisesti turhat bitit poistavasta Isenseen [2004] bitinpakkaustekniikasta, jossa alkuperäisen bittijonon merkitsevät bitit säilyvät ennallaan.

Huffmanin koodaus tuottaa niin sanotun etuliitekoodin, jolla viitataan koodiin, jossa mikään muodostettu koodisana ei ole minkään toisen koodisanan etuliite [Sayood, 2012].

Huffmanin koodissa bittijonot 11 ja 110 eivät molemmat voi siis koskaan vastata jotakin koodattua symbolia, koska 11 on bittijonon 110 etuliite.

Esimerkki Huffmanin koodauksen tuloksesta pakkauksen kohteen ollessa seitsemäntavuinen merkkijono ”aaabbcd” on esitetty kuvassa 8. Kyseisen merkkijonon aakkosto on {a, b, c, d} ja merkkien esiintymistiheys merkkijonossa {0.43, 0.29, 0.14, 0.14}. Esiintymistiheyksiä vastaavista luvuista voidaan käyttää myös nimitystä paino. Huffmanin koodia esittävässä binääripuussa jokaista pakattavaa symbolia vastaa yksi lehtisolmu, mikä on seurausta edellisessä kappaleessa mainitusta etuliitekoodin määritelmästä. Tämä järjestely takaa sen, että mikään symbolia vastaava koodisana ei ole toisen symbolia vastaavan koodisanan etuliite.

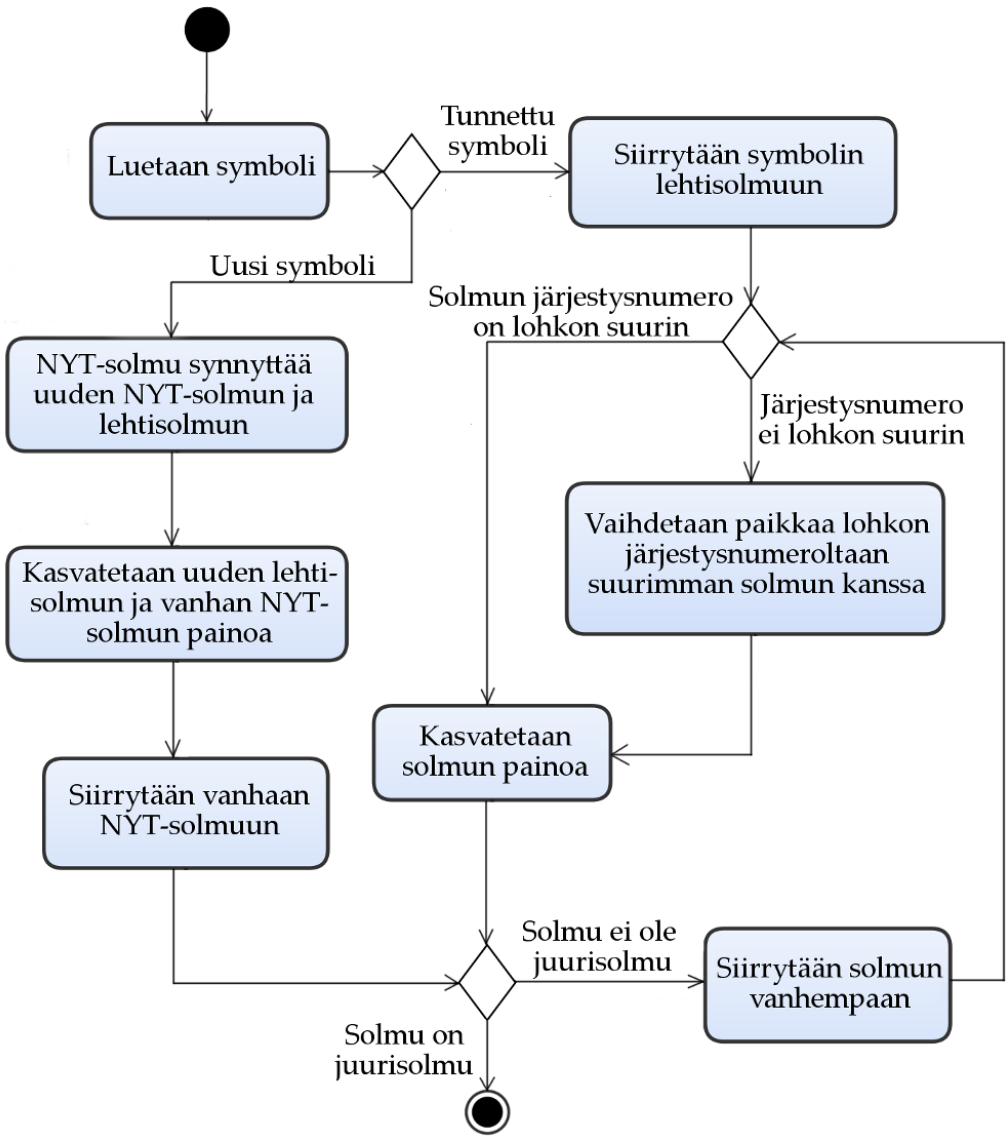


Kuva 8. Huffmanin koodin muodostaminen.

Koodin muodostaminen alkaa kahden harvimmin esiintyvän symbolin sijoittamisesta sisarsolmuiksi, joiden vanhemman paino on kyseisten sisarsolmujen painojen summa. Toisessa vaiheessa luodaan uusi vanhempi, jonka lapsia ovat edellinen vanhempi sekä uusi lehtisolmu, joka vastaa kolmanneksi harvimmin esiintyvää symbolia. Uuden vanhemman painoksi tulee jälleen lapsisolmujen painojen summa. Sama prosessi toistuu, kunnes kaikkia symboleita vastaa jokin binääripuun lehtisolmu. Tällöin puun juurisolmun paino on 1.

Huffmanin koodia purettaessa puuta aletaan lukea juuresta, jolloin kuljettava polku on sitä lyhyempi, mitä yleisempi symboli on kyseessä. Kutakin symbolia vastaava koodisana muodostuu polusta, jota kuljetaan symbolia vastaavalle lehtisolmulle. Koodisanaan eli symbolia vastaavaan bittijonoon lisätään vasemmalle kuljettaessa 0 ja oikealle kuljettaessa 1 [Sayood, 2012]. Kuvan 8 esimerkissä symbolin a koodisana on näin ollen 1, b:n 01, c:n 000 ja d:n 001. Näitä koodisanoja noudattaen alkuperäinen merkkijono ”aaabbcd” voidaan esittää bittijonona `1110101000001`.

Edellä mainittu Huffmanin koodi on pituudeltaan vain 13 bittiä, kun taas alkuperäisen merkkijonon pituus oli peräti seitsemän tavua. On kuitenkin syytä huomioida, että symboleita vastaavien koodisanojen on oltava purkajan tiedossa ennalta, jotta bittijono voidaan purkaa. Muussa tapauksessa koodisanat täytyy toimittaa pakatun tiedon mukana, mikä luonnollisesti lisää tiedonsiirtokustannusta huomattavasti [Sayood, 2012]. Esimerkiksi niin sanottu mukautuva Huffmanin koodaus (eng. adaptive Huffman coding) on tarkoitettu tilanteisiin, joissa koodisanat eivät ole etukäteen purkajan tai pakkaajan tiedossa [Sayood, 2012]. Tällöin koodisanat toimitetaan osana Huffmanin koodia ennalta määritettyä periaatetta käyttäen.



Kuva 9: Mukautuva Huffmanin koodaus, puun päivittäminen. [Sayood, 2012]

Mukautuvan Huffmanin koodin muodostamista on havainnollistettu kuvassa 9. Sayoodin [2012] esittämä mukautuva Huffmanin koodaus käyttää koodin muodostamisessa apuna

niin sanottua NYT (Not Yet Transmitted) -solmua, jonka paino on 0 ja symboli yhteinen sekä lähettäjälle että vastaanottajalle. Mukautuvaa Huffmanin koodausta käytettäessä koodin muodostaja ja purkaja eivät kumpikaan tiedä koodattavien symbolien esiintymistiheyksistä NYT-solmua lukuunottamatta mitään [Sayood, 2012]. Uusi Huffmanin puu koostuu aina yhdestä NYT-solmusta.

Uusina käsitteinä mukautuvaan Huffmanin koodaukseen liittyvät lohko ja solmun järjestysnumero. Lohkolla viitataan kaikkiin puun solmuihin, joilla on sama paino. Järjestysnumero taas on merkitykseltään juuri sellainen kuin voisi olettaakin: mukautuvassa Huffmanin koodauksessa kullakin solmulla on myös oma järjestysnumeronsa puussa. [Sayood, 2012]

Mukautuvassa Huffmanin koodauksessa NYT-solmu lähetetään aina ennen uutta lähettämätöntä symbolia, jotta koodin purkaja osaa seuraavaksi lukea koodisanan sijasta uuden symbolin. Kuten kuvasta 9 voidaan päätellä, tekniikkaan kuuluu oleellisena osana Huffmanin puun päivittäminen solmuja lisättäessä. Kun puu on muodostettu kuvassa 9 esitetyn toimintatavan mukaisesti, tunnettujen symbolien hakeminen koodisanojen perusteella on mahdollista.

Mukautuvan Huffmanin puun periaate on lopulta verrattain yksinkertainen: mikäli symboli ei ole tunnettu, lähetetään NYT-solmu ja uusi symboli. Symbolin ollessa jo tunnettu lähetetään symbolia vastaava koodisana. Kun koodin vastaanottaja kulkee puussa vastaanotetun bittijonon mukaisesti, koodisanan ollessa uusi päädytään aina NYT-solmuun.

Sekä Huffmanin koodauksen että Isenseen [2004] esittämän tekniikan kaltaisia pakkausalgoritmeja voidaan luonnehtia häviöttömiksi pakkaustekniikoiksi, sillä prosessin tuloksena saadaan täysin alkuperäistä arvoltaan vastaava lopputulos. Häviölliset pakkaustekniikat sen sijaan poistavat alkuperäisestä tiedosta sellaiset osat, joiden poisjäännillä ei ole käytännön merkitystä [Sayood, 2012]. Reaaliaikaisten internet-pelien kohdalla yksi esimerkki häviöllisestä pakkauksesta on pelientiteetin nopeutta vastaavan liukuluvun lähettäminen pyöristettynä [van Waveren, 2006]. Pyöristyksestä syntyvän epätarkkuuden voidaan olettaa olevan merkityksetön pelin toiminnan kannalta, mutta silti merkityksellinen säästetyn tilan kannalta.

3.3. Verkkoviiveen torjunta

Erilaiset pakkaustekniikat ovat merkittävä osa tehokkaan verkkoprotokollan toteutusta, mutta reaaliaikaisten internet-pelien kohdalla kenties kriittisin osa-alue on verkkoviiveen torjuminen, johon pakettien koon minimointi ei merkittävästi vaikuta. Pantel ja Wolf [2002] määrittelevät verkkoviiveen aikana, joka kuluu lähetettävän tiedon lähetyksestä sen vastaanottoon. Tätä käsitettä ja sen merkitystä voidaan selventää käytännön esimerkillä.

Oletetaan, että esimerkin internet-peli toimii perinteistä asiakas-palvelin -arkkitehtuuria noudattaen, ja että palvelin vastaa yksin pelin tilan edistämisestä pelaajan syötteiden mukaisesti. Mikäli pelaajan tietokoneen ja palvelimen välinen verkkoviive on esimerkiksi 100 millisekuntia, niin kokonaisverkkoviive pelaajan syötteestä ruudulla näkyvään vasteeseen on 200 millisekuntia. Kokonaisviive on siis kaksinkertainen, sillä pelaajan lähettämän komennon saapumiseen palvelimelle menee ensin 100 millisekuntia, minkä jälkeen palvelimen lähettämän vasteen saapumiseen pelaajalle menee toiset 100 millisekuntia. Esimerkiksi Quaken kaltaisessa pelissä 200 millisekunnin verkkoviive aiheuttaa jo huomattavia ongelmia [Beigbeder *et al.*, 2004; Quax *et al.*, 2004]. On tärkeää ymmärtää myös verkkoviiveen vaikutuksen erot erilaisissa pelitilanteissa. Pelissä esiintyvät tavallista korkeampaa tarkkuutta tai nopeampaa reaktiokykyä vaativat tilanteet ovat huomattavasti kriittisempiä verkkoviiveen suhteen kuin tyypilliset pelitilanteet [Debroy *et al.*, 2013].

Verkkoviiveellä on siis hyvin merkittävä vaikutus reaaliaikaisten verkkopelien tarjoamaan pelikokemukseen – riittävän suuri verkkoviive tekee pelin mielekkäästä pelaamisesta käytännössä mahdotonta. Verkkoviiveen vaikutuksia havainnollistaa hyvin taulukko 2, jossa on kuvattu Pantelin ja Wolfin [2002] tutkimuksessa koottuja subjektiivisia kokemuksia eri suuruisten verkkoviiveiden vaikutuksista pelikokemukseen. Pantelin ja Wolfin [2002] mukaan tutkimuksessa käytettyä testipeliä (Virtual RC Racing) voidaan pitää haastavimpana mahdollisena testiympäristönä pelikokemuksen vaativuuden kannalta. Myöhempien tutkimuksien [Armitage, 2003; Beigbeder *et al.*, 2004; Quax *et al.*, 2004] perusteella voidaan kuitenkin todeta, että esimerkiksi Quaken kaltaiset nopeatempoiset toimintapelit ovat kenties vieläkin vaativampia. Kyseisen lajityypin peleissä jo noin 200 millisekunnin viive aiheuttaa pelaajien poistumista palvelimelta liiallisen verkkoviiveen vuoksi [Zander & Armitage, 2004; Armitage, 2003].

Viive	*Vaikutelma*
500 ms	• pelikokemus ei ole hyväksyttävä • autoa ei pysty ohjaamaan • pelaajan toiminta ja saatu vaste eivät vastaa toisiaan
200 ms	• viive on selvästi havaittavissa • auto on silti ohjattavissa • oman pelityylin sopeuttaminen tilanteeseen on mahdollista • auton kokonaisvaltainen käyttäytyminen ei kuitenkaan ole realistista
100 ms	• hyväksyttävä, mikäli korkeita vaatimuksia realismiin ei ole • viive on huomattavissa pelatessa, mutta ei havaittavissa visuaalisesti
50 ms	• viivettä tuskin huomaa • auton käyttäytyminen vastaa pitkälti viiveetöntä käyttäytymistä

Taulukko 2. Verkkoviiveen vaikutus pelikokemukseen, testiryhmän subjektiivisia havaintoja pelikokemuksesta [Pantel & Wolf, 2002].

Etenkään maantieteellisten etäisyyksien aiheuttamaa verkkoviivettä ei ole nykytekniikalla mahdollista poistaa kokonaan [Pantel & Wolf, 2002]. Verkkoviiveen ollessa näillä näkymin pysyvä osa internet-pelejä sen aiheuttamien ongelmien ratkaisemiseksi on kehitetty alati kasvava kirjo erilaisia tekniikoita [Armitage *et al.*, 2006; Bonham *et al.*, 2000].

Käsitteellä dead reckoning viitataan ekstrapolointitekniikkaan, jolla lasketaan ennuste liikkuvan kohteen nykyisestä sijainnista menneisyydessä olevien sijainti- ja liikeratatietojen perusteella [Diot & Gautier, 1999]. Reaaliaikaisissa internet-peleissä tällaisesta tekniikasta käytetään monesti termiä asiakaspuolen ennuste (eng. client-side prediction) [Armitage *et al.*, 2006; Zander & Armitage, 2004]. Dead reckoning -tekniikkaa käytetään internet-moninpeleissä yleisesti torjumaan verkkoviiveestä ja verkkopakettien katoamisesta aiheutuvia ongelmia [Guo *et al.*, 2003].

Sijaintitietojen ekstrapolointi aiheuttaa kuitenkin ongelmia varsinkin nopeatempoisissa peleissä, joissa pelihahmon nopeus tai suunta saattaa muuttua arvaamattomasti. Mikäli ekstrapoloinnilla laskettu ennuste ei pidä paikkaansa, täytyy pelihahmo siirtää oikeaan paikkaan joko suoraan tai jonkinlaista kompensaatiomenetelmää käyttäen [Stowell *et al.*, 2011; Smed *et al.*, 2002b]. Dead reckoning -periaatteella toimivan sijainnin ennustamis-tekniikan voidaan ajatella koostuvan kahdesta osasta: itse ennustamistekniikasta sekä ennustevirheiden (eng. prediction error) kompensointitekniikasta [Smed *et al.*, 2002b].

Ennustevirheen korjaaminen siirtämällä pelihahmo suoraan todelliseen sijaintiin saattaa aiheuttaa pelihahmojen ”hyppimistä” paikasta toiseen, minkä ei voida ajatella olevan toivottu tilanne.

Pelaaja lähettää tietoa oman pelihahmonsa liikkeistä toisille pelaajille yleensä niin kutsuttua dead reckoning (DR) -vektoria käyttäen [Aggarwal *et al.*, 2004]. DR-vektori sisältää tyypillisesti x-, y- ja z-koordinaatit pelaajan sijainnista sekä pelaajan liikeradan nopeuskomponenttina kullekin koordinaatiston ulottuvuudelle [Aggarwal *et al.*, 2004]. Pelihahmon sijainnin laskettu ennuste perustuu vastaanotettuihin DR-vektoreihin – ilman tietoa pelihahmon viimeisestä vahvistetusta sijainnista, suunnasta ja nopeudesta ennustetta on mahdotonta laskea. Oikeat lähtötiedot eivät kuitenkaan ole tae ennusteen toteutumiselle.

Stowell ja muut [2011] esittävät ekstrapolointitekniikan ongelmaan ratkaisun, jossa sijainnin ekstrapolointi korvataan interpoloinnin ja keinotekoisen viiveen yhdistelmällä. Tällainen tekniikka perustuu pelihahmon siirtämiseen ruudulla vasta, kun kaksi todellista sijaintia on jo tiedossa. Pelaajaa aletaan liikuttaa ensimmäisestä tunnetusta sijainnista toiseen sijaintiin käyttämällä interpolointia tunnettujen sijaintien välissä olevien pisteiden laskemiseen [Stowell *et al.*, 2011]. Jotta kaksi tunnettua sijaintia saadaan muistiin ennen pelihahmon liikuttamista, täytyy pelin keinotekoisesti odottaa palvelinta, kunnes tarvittavat tiedot on vastaanotettu. Tämä tarkoittaa käytännössä sitä, että pelaajan ruudulla näkyvä pelitila on aina myöhässä todellisuudesta, mutta toisaalta pelihahmojen sijaintia ei myöskään tarvitse ennustaa tulevaisuuteen.

Tällainen tekniikka mahdollistaa myös palvelimen tilapäivityspakettien lähetystahdin harventamisen [van Waveren, 2006], sillä pelihahmoja liikutetaan jatkuvasti interpoloimalla nykyinen sijainti. Pelitilanteiden näkyminen myöhässä aiheuttaa kuitenkin muunlaisia ongelmatilanteita, kuten pelaajien keskinäisen vuorovaikutuksen mutkistumisen pelaajien ruudulla näkemien tapahtumien ajoituksen poiketessa toisistaan. Mikäli samaa tekniikkaa sovelletaan myös pelaajan oman pelihahmon liikkeisiin, tekniikka lisää samanlaista viivettä pelituntumaan kuin aikaisemmin kuvattu täysin pelkistetty asiakas-palvelin -malli.

Pelaajan omaa hahmoa voidaankin liikuttaa ekstrapoloimalla pelaajan sijainti paikallisesti, jolloin pelaajan pelituntuma tuntuu käytännössä viiveettömältä. Jotta pelihahmon sijainnin paikallinen ennustaminen onnistuisi tarkasti, täytyy asiakasohjelman sisältää käytännössä sama pelihahmon liikuttamiseen liittyvä logiikka, jolla pelihahmoa liikutetaan myös

palvelimella [Bernier, 2001]. Paras yhdistelmä viiveentorjuntatekniikoita lieneekin pelaajan oman pelaajan liikuttaminen paikallisesti sijaintia ekstrapoloimalla ja muiden pelientiteettien liikuttaminen myöhässä tunnettuihin sijainteihin perustuvalla interpoloinnilla [Bernier, 2001; van Waveren, 2006]. Tämäkään ratkaisu ei kuitenkaan ole ongelmaton, sillä pelaajan omaa pelihahmoa lukuunottamatta pelientiteetit näkyvät pelaajalle myöhässä. Tätä ongelmaa on yritetty ratkaista lisäämällä palvelimelle logiikkaa, joka ottaa huomioon pelaajien ruudulla näkyvän viiveen [Bernier, 2001; van Waveren, 2006]. Tätä tekniikkaa voidaan havainnollistaa esimerkillä.

Kun esimerkiksi Quaken kaltaisessa pelissä paikallinen pelaaja näkee ruudullaan vastustajan ja ampuu tätä kohti, osumaa ei normaalisti tule kohteen myöhässä näkyvän sijainnin takia. Periaatteessa paikallisen pelaajan täytyisikin ampua hieman ruudulla näkyvän kohteen edelle ennakoiden vastustajan reaaliaikainen sijainti nykyiseen liikerataan ja nopeuteen perustuen. Mikäli viivettä on paljon, tämä voi olla haastavaa etenkin nopeatempoisissa peleissä, joissa liikeradat ja nopeudet voivat muuttua nopeasti [Bernier, 2001]. Tällaista tilannetta korjaamaan voidaan käyttää tekniikkaa, joka mahdollistaa kohteeseen osumisen tähtäämällä suoraan ruudulla näkyvää vastustajaa kohti ennakoinnin sijaan [Bernier, 2001; van Waveren, 2006]. Tekniikka perustuu siihen, että palvelin laskee tapahtuneet osumat menneisyyttä vastaavin tiedoin, jolloin laskelmat vastaavat pelaajan ruudulla näkyvää pelitilaa.

Edellä mainitun kaltaisen viivekompensaation varjopuolina ovat melko monimutkaiseksi kasvava palvelinlogiikka sekä tarve tallentaa pelientiteettien vanhat sijainnit osumien laskemista varten [van Waveren, 2007]. Lisätaakkana ovat ongelmatilanteet, joita voi syntyä pelaajien latenssien poiketessa huomattavasti toisistaan. Esimerkiksi matalalatenssinen pelaaja saattaa omalla ruudullaan olla suojassa nurkan takana, kun korkealatenssinen pelaaja ampuu tätä kohti menneisyydessä [Armitage *et al.*, 2006; van Waveren, 2007]. Tällaisessa tapauksessa matalalatenssinen pelaaja saa osuman käytännössä seinän läpi.

Lienee selvää, että erilaiset viiveentorjuntamenetelmät aiheuttavat kaikki omanlaisiansa sivuvaikutuksia, joiden välillä tasapainottelu on haaste verkkokoodin kehittäjille. Paras ratkaisu riippuneekin pitkälti pelin lajityypistä, sillä joissakin tapauksissa toiset sivuvaikutukset voivat olla huomattavasti hyväksyttävämpiä kuin toiset.

3.4. Kokonaiskuva vaatimuksista ja ratkaisuista

Yhteenvetona aihealueesta on luontevaa tarkastella reaaliaikaisten internet-pelien luomia haasteita ja mahdollisia ratkaisuja kokonaisuutena. Taulukko 3 havainnollistaa käsiteltyjä asioita ja sen myötä internet-peliltä verkkoteknisesti vaadittavia ominaisuuksia.

Haaste	***Ratkaisu***
Verkkoviive *Negatiivisten vaikutusten vähentäminen*	• Pelipalvelinten sijoittaminen maantieteellisesti lähelle pelaajia. • Pelientiteettien liikkeiden ennustaminen paikallisesti sijaintien ekstrapoloinnin avulla. • Pelientiteettien liikkeiden näyttäminen myöhässä yhdistettynä sijaintien interpolointiin verkkopakettien välissä.
Verkkoliikenne *Lähetettävän informaation minimointi*	• Sisäiset verkkopaketin pakkausmenetelmät, esimerkiksi bitinpakkausalgoritmit. • Ulkoiset pakkausmenetelmät. Esimerkiksi delta-pakkaus, jossa lähetettävä verkkopaketiin sisällytetään vain tiedot, jotka ovat muuttuneet vastaanottajan viimeksi kuittaaman verkkopaketin suhteen. • Kiinnostuksenhallinta. Lähetettävän tiedon rajaaminen erilaisten algoritmien avulla.

Taulukko 3. Reaaliaikaisten internet-pelien verkkoteknisiä haasteita ja ratkaisuja.

4. Quake-sarjan historiaa

Ennen siirtymistä nykyaikaisten internet-pelien verkkotekniikan yksityiskohtaiseen tarkasteluun on hyvä kerrata hieman aihealueen historiaa. Epäonnistuneiden ja onnistuneiden ratkaisujen dokumentointi käytännön esimerkkien avulla tarjoaa oivan tietopohjan uudempien verkkotekniikkaratkaisujen merkityksen ymmärtämiseksi. Alan pioneerien ensimmäiset yritykset rakentaa reaaliajassa pelattava verkkopeli tarjoavat tässä käytännönläheisen tarkastelukohteen.

Id Softwaren Doom vuodelta 1993 lienee ollut ensimmäinen jaetussa reaaliaikaisessa 3D-virtuaaliympäristössä pelattava moninpeli [Kushner, 2004], joskin sen verkkoarkkitehtuuriratkaisut eivät olleet kelvollisia muuta kuin paikallisessa lähiverkossa tapahtuvaan pelaamiseen. Sen lisäksi, että Doom käytti vertaisverkkomallia verkkoyhteyden toteutuksessa, Doomin verkkoprotokolla perustui luotettavaan tietovirtaan IPX-protokollaa (Internetwork Packet Exchange) hyödyntäen [Id Software, 1997; van Waveren, 2006]. Vaikka Doomin verkkomoninpeli tukikin enintään vain neljää yhtäaikaista pelaajaa [Kushner, 2004], verkkokoodissa tehdyt ratkaisut olisivat tehneet internet-pelaamisen silloisilla tiedonsiirtoyhteyksillä mahdottomaksi. Suhteellisen pienille tietokoneverkoille tarkoitettu IPX-protokolla ei niin ikään olisi ollut tarpeeksi skaalautuva internetin kaltaisille suurille verkoille [Garfinkel *et al.*, 2003].

Id Softwaren vuoden 1996 helmikuussa julkaisema [Kushner, 2004] julkinen Quake-testiversio Qtest avasi internet-pelaamisen jaetussa reaaliaikaisessa 3D-ympäristössä ensi kertaa suurille ihmismassoille [Mäki-Panula, 2001]. Qtestin dokumentaatiota [Romero, 1996] tarkasteltaessa selviää, että Doomin verkkopelistä tutun IPX-protokollan lisäksi Qtest tarjosi erityisesti internet-pelaamiseen tarkoitetun TCP-protokollatuen, joskin tuki modeemiyhteyksille puuttui. Onkin perusteltua olettaa, että luotettavan TCP-protokollan käyttämisen ongelmallisuus ei selvinnyt vielä Qtestin myötä juuri hitaimpien ja samalla yleisimpien verkkoyhteyksien tuen puutteen vuoksi. Kuten Id Softwaren entinen tekninen johtaja John Carmack [1996] myöhemmin QuakeWorld-verkkokoodiin liittyvissä muistiinpanoissaan toteaa, luotettavaan tietovirtaan perustuvan protokollan käyttäminen toimi melko hyvin nopeilla yhteyksillä, mutta hyvin huonosti hitaammilla.

Doomista poiketen Qtest käytti verkkoarkkitehtuuriratkaisunaan asiakas-palvelin -mallia [Romero, 1996], minkä ansiosta ainoastaan palvelimelta vaadittiin tarpeeksi suurta tiedonsiirtokapasiteettia pelin tilan lähettämiseen ja vastaanottamiseen jokaiselta pelaajalta

[Bonham *et al.*, 2000]. Tehdyt muutokset mahdollistivat ensimmäistä kertaa Doomin kaltaisen pelin pelaamisen internetissä, ja muista verkkotekniikkaan liittymättömistä huolenaiheista huolimatta pelaajat olivatkin innoissaan online-”deathmatchin” tuomista mahdollisuuksista [Kushner, 2004]. Deathmatchilla tarkoitetaan 3D-ammuskeluiden perinteisintä moninpelimuotoa, jossa pelaajat koettavat tappaa mahdollisimman monta vastustajaa rajoittaen samalla omien kuolemiensa määrää [Beigbeder *et al.*, 2004].

Qtestistä kerätty informaatio ja sen perusteella tehdyt koodiparannukset joutuivat uuteen koetukseen vuoden 1996 kesäkuussa, kun Quake julkaistiin. Kuvassa 10 kuvankaappaus Qtestin pelikartan test1 alkunäkymästä.

Kuva 10. Qtest, kartta test1.

Quake käytti verkkokoodissaan Qtestin tapaan luotettavaan tietovirtaan perustuvaa lähestymistapaa, joskin Qtestistä poiketen Quakeen oli jälkikäteen lisätty ”sivukaista”, jonka avulla verkkopaketteja voitiin lähettää myös epäluotettavasti [Carmack, 1996; Id Software, 1999]. Quaken verkkoteknisenä perustana oli kuitenkin yhä luotettava protokolla [Carmack, 1996; Id Software, 1999]. Qtestin tulosten myötä Quake tuki Doomin neljän pelaajan sijaan jopa kuuttatoista yhtäaikaista pelaajaa verkon yli pelattaessa [Kushner, 2004]. Kuten Carmackin [1996] muistiinpanoista ja myöhempien muutosten perusteella selviää, Quaken verkkokoodi oli kuitenkin vielä varsin virheherkkä ja perusoletuksiltaan kyseenalainen.

Carmackin [1996] mukaan erityisesti hitaiden modeemiyhteyksien aiheuttamat ongelmat johtivat Quaken alkuperäisen verkkokoodin hylkäämiseen ja uuden, epäluotettavaan UDP-protokollaan perustuvan verkkokoodin luomiseen. Joulukuussa 1996 Id Software julkaisi Quakeen ilmaisen QuakeWorld-päivityksen, jota voitaneen pelin verkkokoodin perusratkaisuilta ja -oletuksilta pitää ensimmäisenä "nykyaikaisena" reaaliaikaisena internet-pelinä. QuakeWorld toteutti ensimmäistä kertaa sekä asiakas-palvelin -mallin mukaisen verkkoarkkitehtuurin että epäluotettavaan tiedonsiirtoon perustuvan räätälöidyn verkkoprotokollan [Id Software, 1999]. Lisäksi QuakeWorld-asiakasohjelma käytti *dead reckoning* -tyyppistä ekstrapolointitekniikkaa pelaajien sijainnin ennustamiseen kohdassa 3.3. havainnollistettuun tapaan [Id Software, 1999].

Vuoden 1997 joulukuussa julkaistu Quake 2 pohjautui verkkoteknisesti pitkälti QuakeWorldin teknologiaan, eikä suuria muutoksia QuakeWorldiin ole nähtävissä lähdekoodia [Id Software, 1999; 2001] tarkasteltaessa. Vuoden 1999 Quake 3 sen sijaan sisälsi merkittäviä innovaatioita ja muutoksia Quake-protokollan toimintatavassa [Id Software, 2005], ja juuri Quake 3:n verkkotekniset ratkaisut ovatkin oiva aihe lähempään tarkasteluun lähdekoodianalyysin avulla.

Myös vuonna 2004 julkaistun Doom 3:n lähdekoodi [Id Software, 2011] on tarkasteltavissa, mutta uudemmuudestaan huolimatta Doom 3 ei verkkoteknisesti ole Quake 3:a kiinnostavampi tarkastelun kohde. Doom 3 ei koskaan ollut pääasiallisesti suunnattu verkkopelaamiseen, mistä kielii muun muassa alkuperäisversion tuki vain neljälle samanaikaiselle pelaajalle [Accardo, 2004; Adams, 2004]. Heikommasta asemasta kertoo myös Doom 3:n ja Doom 3:een pohjautuvan Quake 4:n nykyinen palvelintarjonta: GameTracker-sivuston [2014] palvelinhakukoneen mukaan Doom 3 -palvelimia on hieman yli kymmenen ja Quake 4 -palvelimia hieman alle 30. Quake 3 -palvelimia sen sijaan on yhä lähes 400, minkä lisäksi QuakeWorldin ja Quake 2:n palvelimia on kuitakin enemmän kuin Doom 3 -ja Quake 4 -palvelimia yhteensä. Lisäksi vuonna 2009 julkaistu suosittu Quake Live [Id Software, 2014] pohjautuu Quake 3:n tekniikkaan [Sanglard, 2011; Maiberg, 2014].

Uusimpien reaaliaikaisten internet-pelien lähdekoodin tarkastelu ei lähtökohtaisesti ole mahdollista, ja Id Softwaren politiikka julkaista pelien lähdekoodit vapaaseen levitykseen onkin nähtävä poikkeuksellisena. Quake 3:n verkkotekniikan lähempi tarkastelu lienee siis perusteltua sekä sen vakiintuneisuuden että avoimen lähdekoodin perusteella. Alkuperäisen Quake 3:n lähdekoodin [Id Software, 2005] sijaan voidaan tarkasteluun ottaa

kolmansien osapuolien monella osa-alueella kehittämä Quake 3 -versio ioquake3 [The ioquake Group, 2014].

Kuten aikaisemmilla Quake-sarjan verkkoprotokollille, myös ioquake3:n protokollalle on annettu oma versionumero. Samaan tekniikkaan perustuvat verkkoprotokollat ymmärtävät samankaltaisia tai jopa samoja komentoja, joten protokollien erottaminen toisistaan versionumeroinnilla on välttämätöntä. Versionumeroiden kehitystä vuosien saatossa on havainnollistettu taulukossa 4. On oleellista tiedostaa, että vaikka protokolla 70 perustuu vahvasti vanhempiin protokollaversioihin, sen toimintaperiaatteet saattavat erota merkittävästi vanhemmista versioista.

Peli	***Protokollaversio***
Quake	15 [Id Software, 1999]
QuakeWorld	28 [Id Software, 1999]
Quake 2	34 [Id Software, 2001]
Quake 3	68 [Id Software, 2005]
ioquake3	70 [The ioquake Group, 2014]

Taulukko 4. Quake-sarjan verkkoprotokollien versiointi.

5. Ioquake3-protokollan keskeiset ominaisuudet

Aloitetaan ioquake3:n verkkoprotokollan tarkastelu yleiskatsauksella protokollan keskeisimpiin ominaisuuksiin, minkä jälkeen siirrytään yksityiskohtaisempaan lähdekooditason tarkasteluun.

5.1. Käytetyt verkkotekniset ratkaisut

Ioquake3-protokolla on UDP/IP-protokollan päälle rakennettu sovelluskerroksen protokolla, johon on lisätty muun muassa luotettavuus-, vuonohjaus- ja ruuhkanhallintaominaisuuksia. Lisäksi ioquake3:n verkkokoodissa on toteutettu monia luvussa 3 käsiteltyjä verkkopeleille tyypillisiä tekniikoita. Reaaliaikaiselle internet-pelille tyypillisesti suurin osa liikenteestä on epäluotettavaa, ja käytännössä luotettavia viestejä lähetetäänkin vain poikkeustapauksissa. UDP-protokollasta poiketen ioquake3:n verkkoprotokolla on yhteydellinen protokolla, joskin lähinnä yhteyden muodostamista varten protokolla tukee myös joitakin yhteydettömästä tilasta lähetettyjä komentoja. Verkkokoodin keskeisiä teknisiä ominaisuuksia on havainnollistettu taulukossa 5.

Haaste	***Ratkaisu***
Verkkoliikenne	• Delta-pakkaus. • Verkkopaketin dynaaminen viestirakenne. • Huffmanin koodaus. • Kiinnostuksenhallinta PVS:n perusteella.
Verkkoviive	• Pelientiteettien sijaintien laskeminen tapauskohtaisesti joko interpolointi- tai ekstrapolointitekniikalla. • Asiakas-palvelin -arkkitehtuuri.
Virheenhallinta	• Valikoiva luotettavuus. • Ruuhkanhallinta, vuonohjaus. • Haaste-vaste -autentikointimenetelmä.

Taulukko 5. Ioquake3:n verkkokoodin teknisiä ominaisuuksia. Johdettu ioquake3:n lähdekoodista [The ioquake Group, 2014].

Taulukon 5 mukaisen todistetusti toimivan kokonaisuuden käytännön toteutukseen voidaan perehtyä tutkimalla oleellisimmissa verkkokoodin funktioissa käytettyjä ohjelmointiteknisiä ratkaisuja. Yksityiskohtainen tarkastelu on pyritty kohdistamaan niihin ioquake3:n osa-alueisiin, jotka ovat tälle protokollalle ominaisia, toteutukseltaan mielenkiintoisia ja lisäksi sovellettavissa muihin reaaliaikaisiin internet-peleihin.

Joidenkin käsiteltävien osa-alueiden lähdekooditason toteutuksessa saattaa olla kääntöympäristöstä riippuvia alustakohtaisia eroja. Alustakohtaiset erot on tarvittaessa huomioitu erillisellä maininnalla, mutta toiminnallisuutta pyritään lähtökohtaisesti kuvaamaan lähdekoodin kääntöympäristöstä riippumattomalla tavalla. Ellei toisin ole erikseen mainittu, kaikki lähdekooditiedostot, johon tässä luvussa viitataan, ovat ladattavissa ioquake3-projektin git-lähdekoodirepositoriosta [The ioquake Group, 2014].

Ioquake3:n protokollan toiminnallisuuteen syventyminen on luontevaa aloittaa tiedonsiirron perusyksikön eli verkkopaketin rakenteesta. Verkkopaketin rakenteella on oma merkityksensä useassa ioquake3:n protokollan korkeamman tason ominaisuudessa.

5.2. Verkkopaketin rakenne

Kuten kohdan 2.3. esimerkkisovellus, myös ioquake3 lisää protokollapinon sovelluskerroksessa verkkopakettiin oman otsikon ja viestiosion. Protokolla 70:n mukaiset verkkopaketit alkavat yhteydettömästä tilasta lähetettyjä paketteja lukuunottamatta aina 10-tavuisella otsikolla, jonka sisältö on kuvattu taulukossa 6. Verkkopaketin otsikko ei eroa alkuperäisestä Quake 3:n protokollasta [Id Software, 2005].

Kuten taulukosta 6 voidaan päätellä, ioquake3:n verkkoprotokolla osaa TCP-protokollan tapaan jakaa isot verkkopaketit sirpaleiksi, mikä onkin oleellinen ominaisuus verkkoprotokollan tehokkuuden kannalta. Ioquake3:n lähdekooditiedostossa net_chan.c verkkopaketin maksimikooksi määritetään 1400 tavua – tätä isommat paketit jaetaan sirpaleiksi ja kootaan takaisin yhteen vastaanottajalla kohdassa 2.3. mainittuun tapaan. Ottaen huomioon protokollapinon alempien protokollien aiheuttama yleiskustannus 1400 tavun raja juontunee ethernet-protokollan verkkopakettien 1500 tavun maksimikoosta (MTU) [Stevens *et al.*, 2003]. Sirpaloinnin tekeminen sovelluskerroksen protokollassa mahdollistaa sirpaloinnin hallinnan toteuttamisen sovellukselle optimaalisella tavalla.

Toisin kuin kuljetuskerroksen UDP-protokolla, ioquake3:n protokolla pitää kirjaa lähetetyistä ja vastaanotetuista paketeista järjestysnumeroiden avulla. Kuten Stevens ja muut [2003] toteavat, sovelluskerroksen protokollaan luotettavuutta lisätessä järjestysnumerot ovat välttämätömiä. Aikaisemmista Quake-sarjan peleistä [Id Software, 1999; 2001] poiketen ioquake3:ssa vain lähtevän paketin järjestysnumero on kiinteä osa verkkopaketin otsikkoa. Muut tarvittavat tiedot muun muassa luotettavuuden toteuttamiseen lähetetään verkkopaketin viestiosiossa. Järjestysnumeroiden siirtäminen viestiosioon liittynee verkkopakettien sirpalointiin, sillä kaikkia järjestysnumeroita ei ole

tarpeen lähettää jokaisen sirpaleen mukana. QuakeWorld [Id Software, 1999] ja Quake 2 [Id Software, 2001] eivät sisällä toiminnallisuutta verkkopakettien sirpalointiin sovelluskerroksessa.

Pituus (tavua)	***Sisältö***	***Tarkoitus***
4	Lähtevän paketin järjestysnumero	Yksilöi paketin juoksevalla järjestysnumerolla. Kentän merkitsevin bitti kertoo myös, onko kyseessä sirpaloitu viesti. Sirpaloiduilla viesteillä on sama järjestysnumero. Mikäli järjestysnumero on -1, paketti käsitellään kaistan ulkopuolisena liikenteenä.
2	Asiakasohjelman yksilöllinen portti	Asiakasohjelman määrittämä sattumanvarainen porttinumero, qport. Yksilöi asiakasohjelman. Palvelimen lähettämät paketit eivät sisällä tätä kenttää.
2	Sirpaleen aloitustavu	Kertoo verkkopaketin sisältämän sirpaleen sijainnin sirpaloidussa viestissä. Käytössä vain sirpaloiduissa verkkopaketeissa.
2	Sirpaleen pituus	Kertoo verkkopaketin sisältämän sirpaleen pituuden. Käytössä vain sirpaloiduissa verkkopaketeissa.

Taulukko 6: Ioquake3-verkkopaketin otsikko. Johdettu lähdekooditiedostosta net_chan.c [The ioquake Group, 2014].

Järjestysnumeron lisäksi verkkopaketin otsikossa on kolme kaksitavuista kenttää, joista ensimmäinen on niin sanottu qport. Kuten net_chan.c-lähdekooditiedoston kommenteista selviää, qport on varotoimenpide joidenkin reitittimien epätoivottuun tapaan vaihtaa verkkopakettien UDP-lähdeporttia pelin aikana. Qport-numeron ja ip-osoitteen avulla asiakasohjelma voidaan yksilöidä pitävästi, vaikka UDP-porttinumero muuttuisi.

Otsikon viimeiset neljä tavua sisältävät tarpeelliset tiedot sirpaloitujen verkkopakettien kokoamiseen vastaanottajan päässä muodostaen näin otsikon kokonaiskooksi korkeintaan 10 tavua. Ioquake3:n käyttämän protokollapinon yleiskustannus paketin lähettämiseen on näin ollen 58 tavua, joista 8 tavua on UDP-protokollan [Comer, 2000], 20 tavua

IP-protokollan ja 20 tavua ethernet-protokollan [Stevens, 1993] aiheuttamaa yleiskustannusta.

Verkkopaketin varsinainen viestiosuus alkaa otsikon jälkeen. Viestiosuus noudattaa asiakkaan palvelimelle lähettämissä verkkopaketeissa tyypillisesti taulukossa 7 esitettyä rakennetta. Viestin dynaaminen osio eli kaksi viimeistä kenttää koostuvat tyypillisesti komentojen lukumäärästä, komentojen tunnisteesta ja parametreista. Palvelin tulkitsee komentojen parametrit komennon tunnisteen perusteella.

Pituus (tavua)	***Sisältö***	***Tarkoitus***
4	Palvelimen tilatunniste	Palvelimen määräämä tunniste, joka vaihtuu muun muassa pelikartan vaihtuessa. Vanhentuneilla tilatunnisteilla saapuvia viestejä ei suoriteta, vaan asiakkaalle lähetetään täysi tilapäivitys ja uusi tunniste.
4	Vastaanotetun paketin järjestysnumero	Viimeisimmän palvelimelta vastaanotetun verkkopaketin järjestysnumero.
4	Vastaanotetun luotettavan viestin järjestysnumero	Viimeisimmän palvelimelta vastaanotetun luotettavan viestin järjestysnumero.
n	Mahdolliset luotettavat asiakaskomennot	Mahdolliset asiakkaan palvelimelle lähettämät luotettavat komennot.
n	Muut komennot	Tavalliset epäluotettavat komennot, kuten pelaajan liikkuminen.

Taulukko 7: Tyypillisen asiakkaan palvelimelle lähettämän ioquake3-verkkopaketin viestiosio. Johdettu lähdekooditiedostosta cl_input.c [The ioquake Group, 2014].

Palvelimen asiakkaalle lähettämät verkkopaketit noudattavat samantapaista rakennetta, joskin normaalissa pelitilanteessa pakettiin sisältyy aina myös palvelimen pelitilan tilannevedos, jota käytetään muun muassa delta-pakkaukseen. Lisäksi palvelimen tilatunnisteen sijasta lähetetään palvelimen aikaleima, jota käytetään muun muassa pelitilan paikallisen simuloinnin ekstrapolointi- ja interpolointilaskuihin sekä tilannevedosten järjestämiseen aikajärjestykseen.

5.3. Verkkokanavatietueen rakenne

Ioquake3:n verkkokanavaa voidaan pitää protokolla 70:n tapana toteuttaa muun muassa yhteydellisyyden ja luotettavuuden elementit, jotka puuttuvat UDP-protokollasta. Yhteydettömästä tilasta lähetettyjä viestejä lukuunottamatta kaikki ioquake3:n lähettämä liikenne kulkee asiakaskohtaisen verkkokanavan kautta. Jokainen yhteydellisestä tilasta lähetetty paketti sisältää ioquake3-protokollan mukaisen otsikon ja viestiosion sekvenssitiedot viimeksi lähetetyistä ja vastaanotetuista paketeista sekä viesteistä.

Tyyppi	***Kentän nimi***	***Tarkoitus***
int	incomingSequence	Viimeksi vastaanotetun paketin järjestysnumero.
int	outgoingSequence	Viimeksi lähetetyn paketin järjestysnumero.
int	fragmentSequence	Viimeksi vastaanotetun sirpaleen järjestysnumero.
int	fragmentLength	Viimeksi vastaanotetun sirpaleen pituus tavuina.
byte	fragmentBuffer	Vastaanotettujen sirpaleiden tallennuspuskuri. Käytetään sirpaleiden kokoamiseen.
qboolean	unsentFragments	Kertoo, onko lähetysjonossa jäljellä lähettämättömiä sirpaleita. Tosi/epätosi.
int	unsentFragmentStart	Lähetysjonon seuraavan sirpaleen alkusijainti.
int	unsentLength	Lähetysjonossa jäljellä olevien sirpaleiden kokonais-pituus tavuina.
byte	unsentBuffer	Sirpaleiden lähetyspuskuri.

Taulukko 8. Netchan_t-tietueen oleellisimmat kentät. Johdettu lähdekooditiedostosta qcommon/qcommon.h [The ioquake Group, 2014].

Suuri osa ioquake3:n verkkokanavan toimintaan tarvittavista tiedoista tallennetaan net.h-otsikkotiedostossa esiteltyyn netchan_t-tyyppiseen tietueeseen, jollainen luodaan jokaiselle palvelimelle yhdistävälle asiakkaalle. Taulukkossa 7 on esitetty verkkokanavatietueen tärkeimmät kentät ja niiden tarkoitus. Asiakkaan ja palvelimen verkkokanavatietueet ovat rakenteeltaan samanlaiset. Taulukon 7 kenttien lisäksi tietueessa pidetään kirjaa lisäksi muun muassa kadonneiden pakettien lukumäärästä sekä latenssitiedoista.

Taulukossa 7 esitetyt kentät koskevat verkkopakettien sekvenssinumeroiden lisäksi lähinnä sirpaloitujen verkkopakettien käsittelyyn liittyviä tietoja. Varsinaisen verkkokanavatietueen lisäksi oleellisia tietueita ovat myös client.h-otsikkotiedoston clientConnection_t sekä server.h-otsikkotiedoston client_t. Näihin tietueisiin tallennetaan tiedot muun muassa vastaanotetuista ja lähetetyistä luotettavista komennoista sekä niiden kuittauksista. Yhdessä nämä tietueet mahdollistavat muun muassa protokollan valikoivan luotettavuuden.

5.4. Verkkoliikennöinti

Sovelluskerroksen verkkokoodin alimmalla tasolla verkkopakettien vastaanottoon ja lähetykseen käytetään käyttöjärjestelmän kirjastoihin kuuluvia UDP-protokollan funktiota. UDP-protokollaa käyttävien funktioiden kutsuminen voidaan mieltää siirtymiseksi sovelluskerroksen protokollasta kuljetuskerrokseen. Tietoa lähetettäessä kuljetuskerroksen kutsuminen on funktioketjun viimeinen funktio, kun taas tietoa vastaanottaessa tilanne on päinvastainen.

Ennen verkkopakettien lähetykseen ja vastaanottoon käytettävien kuljetuskerroksen funktioiden käyttöä täytyy verkkoyhteys alustaa. Tämä alkaa niin sanotun UDP-pistokkeen avaamisella, joka tarjoaa rajapinnan kuljetuskerroksen UDP-protokollan käyttöön. Tämä toimenpide on rutiininomainen mille tahansa UDP-protokollaa käyttävälle verkkosovellukselle, joten kyseisen prosessin kuvaamiseen riittää tässä pinnallinen tarkastelu.

Verkkoyhteyden alustukseen liittyvä toiminnallisuus keskittyy ioquake3:ssa net_ip.c-lähdekooditiedostoon. Käytettävä UDP-pistoke avataan käyttöjärjestelmän C-kirjastoihin kuuluvalla funktiolla *socket()*, minkä jälkeen avattu pistoke sidotaan paikalliseen IP-osoitteeseen ja ioquake3:n käyttämään porttiin funktiolla *bind().* Edellä esitettyjen vaiheiden lisäksi lähdekoodi sisältää huomattavan määrän erilaisia virheentarkistuksia, joilla varmistetaan pistokkeen avaamisen onnistuminen. Sivuhuomiona mainittakoon, että Unix-pohjaisissa käyttöjärjestelmissä *socket()* palauttaa Windowsin pistokedeskriptoria geneerisemmän tiedostodeskriptorin, joka tukee myös tiedostonhallintaan tarkoitettuja funktioita [Yau & Lam, 1998]. Tämä on syytä ottaa huomioon, kun samaa lähdekoodia on tarkoitus käyttää eri alustoilla.

Verkkoyhteyden alustuksen jälkeen yhteyttä voidaan käyttää tiedon lähetykseen yhteydettömästä tilasta, mutta verkkokanavan tarjoama ylemmän tason toiminnallisuus ei

ole vielä käytettävissä. Jotta sovelluskerroksen protokollan täysi toiminnallisuus saadaan käyttöön, täytyy UDP-yhteyden lisäksi avata ioquake3:n verkkokanava. Ennen verkkokanavan avausta protokolla tukee ainoastaan yhteydettömästä tilasta lähetettyjä viestejä.

Kuten kohdassa 5.2. mainittiin, yhteydettömästä tilasta tai vastaavasti kaistan ulkopuolisena liikenteenä lähetetyt verkkopaketit ovat poikkeus aiemmin esitettyyn verkkopaketin rakenteen määritelmään. Tällaisten verkkopakettien muodostaminen ja käsittely poikkeaa tavallisista ioquake3-verkkopaketeista selvästi. Tarkastellaan seuraavaksi verkkokanavan ulkopuolisten ja yhteydettömästä tilasta lähetettyjen verkkopakettien käsittelyä.

Oleellisin yhteydettömästä tilasta tapahtuvaan tiedonsiirtoon liittyvä funktio on *NET_OutOfBandPrint()*. Kuten funktion nimestä saattaa päätellä, samaa funktiota käytetään myös kaistan ulkopuolisten viestien lähettämiseen, kun verkkokanava on jo avattu. Yhteydettömästä tilasta lähetettyjen ja kaistan ulkopuolisten verkkopakettien välinen ero onkin ioquake3:n protokollassa ainoastaan käsitteellinen, sillä pakettien muodostamis- ja käsittelytapa on kummassakin tilanteessa sama.

Ioquake3 ei lähetä kaistan ulkopuolista liikennettä kuin muutamissa erikoistapauksissa. Esimerkiksi yhteyden muodostamiseen tai yhteyden tilan muutoksiin liittyvät viestit lähetetään aina kaistan ulkopuolisina viesteinä, joskin toiminnallisuus on sama riippumatta siitä, onko yhteydellinen tila aikaisemmin muodostettu vai ei. Kaistan ulkopuolisten viestien lähetykseen ei siis tässä tapauksessa käytetä mitään aiemmin muodostetun yhteydellisen tilan ja avatun verkkokanavan mahdollistamaa toiminnallisuutta.

Yhteyden muodostamiseen liittyvien tilanteiden lisäksi ioquake3 käyttää kaistan ulkopuolista liikennettä viestittämään tietoja esimerkiksi käynnissä olevasta tiedostonsiirrosta. Tieto siirron edistymisestä on lähetettävä kaistan ulkopuolella, koska pelin pääasiallinen tiedonsiirtoyhteys toimii tiedostonlähetyksissä TCP-protokollan tapaisena luotettavana tietovirtana, jota ei voida häiritä lisäämällä tietovirtaan siirrettävään tiedostoon kuulumattomia paketteja. Niin ikään palvelimen etähallintaan käytettävät etäkonsolikomennot lähetetään kaistan ulkopuolisena liikenteenä, jotta palvelinta hallitakseen ylläpitäjän ei tarvitse liittyä peliin.

Yhteydettömästä tilasta tai kaistan ulkopuolisena liikenteenä lähetettyihin viesteihin ei lisätä kohdassa 5.2. esitettyä verkkopaketin otsikkoa, vaan ne sisältävät sen sijaan neljätavuisen tunnisteen, jossa kaikki bitit ovat päällä. Jonkinlaisen tunnisteen käyttäminen on välttämätöntä, jotta käytetyn protokollan mukaiset verkkopaketit voidaan erottaa muusta verkkoliikenteestä. Tämä on oleellista varsinkin yhdistysvaiheessa, jossa mahdolliset asiakkaat ovat vielä tuntemattomia. Tunniste lisätään verkkopakettiin *NET_OutOfBandPrint()*-funktiossa koodikatkelmassa 1 kuvattuun tapaan. Tunnisteen lisäämisen jälkeen paketti lähetetään funktiolla *NET_SendPacket()*.

```
1  void QDECL NET_OutOfBandPrint( netsrc_t sock, netadr_t adr, const char
2     *format, ... ) {
3     ...
4     char        string[MAX_MSGLEN];
5     // set the header
6     string[0] = -1;
7     string[1] = -1;
8     string[2] = -1;
9     string[3] = -1;
10    ...
11 }
```

Koodikatkelma 1. Funktio net_chan.c : Net_outOfBandPrint() [The ioquake Group, 2014].

Neljätavuinen tunniste muodostetaan koodikatkelmassa 1 käyttämällä taulukkoa, jonka kantatyyppi on char. Char-tietotyypin etumerkkisyyden oletusasetus riippuu järjestelmäkohtaisesta C-otsikkotiedostosta limits.h, minkä ansiosta etumerkkisyyden määrittämättä jättäminen saattaa joissakin tilanteissa tuottaa ennalta-arvaamattomia tuloksia [Kernighan & Ritchie, 1988]. Tietotyypin mahdollisella etumerkittömyydellä ei ole kuitenkaan tässä tapauksessa merkitystä, sillä alakohdassa 2.5.2. mainittuun tapaan arvon -1 sijoittaminen muuttujaan tuottaa joka tapauksessa bittijonon, joka vastaa kahden komplementtijärjestelmässä arvoa -1. Vain säilöttävällä bittijonolla on tässä tapauksessa merkitystä, ja kuten alakohdassa 2.5.2. esitettiin, arvo -1 vastaa kahden komplementtijärjestelmässä bittijonoa, jossa kaikki bitit ovat päällä. Vastaava bittijono voitaisiin tuottaa kenties selkeämmin heksadesimaaleja käyttäen: merkkijono `"\xff\xff\xff\xff"` tuottaa saman lopputuloksen, ja tällaisen merkkijonon muodostamiseen ja muuttujaan sijoittamiseen tarvitaan ainoastaan yksi koodirivi.

Tarkastellaan seuraavaksi yhteydettömästä tilasta lähetettyjen viestien käsittelyä käytännössä. Tätä voidaan havainnollistaa käymällä läpi verkkokanavan avaamiseen johtava yhteydenmuodostusprosessi.

5.4.1. Verkkokanavan avaus

Kuten aiemmin mainittiin, yhteydenmuodostusprosessi tähtää ioquake3:n protokollan ominaisuuksien täyteen käyttöönottoon. Koska julkiset ioquake3-palvelimet hyväksyvät tavallisesti verkkopaketteja mistä tahansa IP-osoitteesta, täytyy pelin verkkoprotokollan pystyä erottamaan haluttu verkkoliikenne sattumanvaraisesta, kenties jopa haitallisesta liikenteestä. Ioquake3:n eräänlaista haaste-vaste -menetelmää käyttävä yhteydenmuodostusprosessi auttaa osaltaan tässä, sillä yhteydelliseen tilaan siirrytään vasta, kun palvelin ja asiakas ovat varmistuneet verkkoprotokollien yhteensopivuudesta.

Yhteydenmuodostusprosessi alkaa, kun käyttäjä antaa sovellukselle käskyn yhdistää tietyssä IP-osoitteessa sijaitsevaan palvelimeen. Tällöin asiakasohjelmassa suoritetaan cl_main.c-lähdekooditiedoston funktio *CL_Connect_f()*, joka asettaa verkkoyhteyden tilaksi CA_CONNECTING sekä luo niin kutsutun haastenumeron (eng. challenge). Tästä asiakasohjelman muodostamasta sattumanvaraisesta numerosta voidaan käyttää nimitystä asiakkaan haastenumero. Yhdistämisprosessi jatkuu funktion *CL_CheckForResend()* switch-lausekkeessa, jossa edetään verkkoyhteyden tilan perusteella yhteyspyyntöviestin muodostamiseen. Tämä vaihe on kuvattu koodikatkelmassa 2.

```
1  void CL_CheckForResend( void ) {
2  {
3     case CA_CONNECTING:
4     ...
5       // The challenge request shall be followed by a client challenge so
6       // no malicious server can hijack this connection.
7       // Add the heartbeat gamename so the server knows we're running the
8       // correct game and can reject the client with a meaningful message
9       Com_sprintf(data, sizeof(data), "getchallenge %d %s", clc.challenge,
10          Cvar_VariableString("sv_heartbeat"));
11
12      NET_OutOfBandPrint(NS_CLIENT, clc.serverAddress, "%s", data);
13      break;
14    ...
15 }
```

Koodikatkelma 2. Funktio cl_main.c : CL_CheckForResend() [The ioquake Group, 2014].

Koodikatkelmasta 2 nähdään, että kun verkkoyhteys on tilassa CA_CONNECTING, lähetetään etäpalvelimelle riveillä 9-10 kuvattu viesti. Viesti koostuu komennosta ”getchallenge”, jonka parametreina annetaan asiakkaan haastenumero ja pelin sisäisen muuttujan sv_heartbeat arvo. Ioquake3:ssa tämän muuttujan arvo on normaalisti ”OpenArena-1”. Kuten koodikatkelman kommenteista on pääteltävissä, ”OpenArena-1”-merkkijonon lisääminen on varotoimi, jolla vältetään sekaannus muiden Quake-sarjan verkkoprotokollaan pohjautuvien pelien ja pelimodifikaatioiden kanssa.

Komento ”getchallenge” on nimittäin ollut käytössä jo QuakeWorldin verkkoprotokollasta lähtien [Id Software, 1999]. Muodostetun viestin edelle lisätään aiemmin tässä alakohdassa kuvattu neljätavuinen tunniste, minkä jälkeen verkkopaketti lähetetään palvelimelle. Ioquake3-palvelin käsittelee sisääntulevan liikenteen sv_main.c-tiedoston funktiossa *SV_PacketEvent()*, joka on oleellisilta osin esitetty koodikatkelmassa 3.

```
1  void SV_PacketEvent( netadr_t from, msg_t *msg ) {
2  {
3      ...
4      // check for connectionless packet (0xffffffff) first
5      if ( msg->cursize >= 4 && *(int *)msg->data == -1) {
6          SV_ConnectionlessPacket( from, msg );
7          return;
8      }
9      ...
10 }
```

Koodikatkelma 3. Funktio sv_main.c : SV_PacketEvent() [The ioquake Group, 2014].

SV_PacketEvent() käsittelee yhteydettömästä tilasta lähetetyt paketit erillisellä ehtolausekkeella ennen normaalien pakettien lukemista. Tämä kielii myös kohdassa 2.4. mainitusta kaistan ulkopuolisen liikenteen kiireellisyydestä, sillä kuten tässä alakohdassa aiemmin mainittiin, kaistan ulkopuolinen liikenne käsitellään ioquake3:ssa täsmälleen samoin kuin normaalit yhteydettömästä tilasta lähetetyt viestit. Koodikatkelmassa 3 tämä tehdään rivillä 5, jonka tarkasteleminen lähemmin lienee paikallaan.

Int-kokonaislukutyyppi on C-kielessä joko 16-bittinen tai 32-bittinen riippuen kääntäjästä ja laitteistosta [Kernighan & Ritchie, 1988]. Yleensä int-tietotyypin koon määrää laitteiston luonnollinen operandikoko [Kernighan & Ritchie, 1988], joka moderneissa 32- ja 64-bittisissä suoritinarkkitehtuureissa on 32 bittiä eli neljä tavua [AMD, 2013]. Koska yhteydettömästä tilasta lähetetyn verkkopaketin viesti alkaa niin ikään neljätavuisella tunnisteella, se voidaan lukea kätevästi kerralla käyttäen tyyppimuunnosta int-kokonaislukutyyppiin ja vertailuarvoa -1 alakohdan 2.5.2. periaatteen mukaisesti.

Mikäli bittijono vastaa yhteydettömästä tilasta lähetetyn paketin tunnistetta, suoritetaan sv_main.c-lähdekooditiedoston funktio *SV_ConnectionlessPacket()*. Tämä funktio lukee verkkopaketin sisältämän komennon ”getchallenge” ja suorittaa asiaankuuluvan funktion, joka on tässä tapauksessa *SV_GetChallenge()*. *SV_GetChallenge()* muodostaa uuden sattumanvaraisen haastenumeron ja lähettää asiakkaalle yhteydettömästä tilasta komennon ”challengeResponse”. Komennon parametreina lähetetään vastaanotetun ”getchallenge”-komennon parametrina saatu asiakkaan haastenumero, nyt muodostettu

palvelimen haastenumero sekä protokollan versionumero edellä mainitussa järjestyksessä. Palvelimen haastenumero on IP-osoitekohtainen, joten sitä voi käyttää vain se asiakas, jolle haastenumero on alun perin osoitettu.

Yhteyspyyntöviestin vastaus käsitellään asiakasohjelman *CL_PacketEvent()*-funktiossa täsmälleen samaan tapaan kuin palvelimella koodikatkelmassa 3. Yhteydettömästä tilasta lähetetyt paketit käsittelee asiakasohjelmassa funktio *CL_ConnectionlessPacket()*, joka vastaa palvelimen funktiota *SV_ConnectionlessPacket()*. Vastaanotetun komennon ollessa ”challengeResponse” asiakasohjelma varmistaa vastausviestin kelvollisuuden vertailemalla vastaanotetun komennon parametreja paikallisiin tietoihin. Asiakasohjelman aiemmin muodostaman haastenumeron on täsmättävä palvelimen vastausviestissä olevan asiakkaan haastenumeron kanssa, minkä lisäksi protokollan versionumeroiden on täsmättävä. Mikäli tarkistukset onnistuvat, myös vastausviestin toinen parametri eli palvelimen haastenumero kirjataan muistiin ja verkkoyhteyden tilaksi astetaan CA_CHALLENGING.

Yhteyden tilan muututtua asiakasohjelma muodostaa jälleen uuden yhteydettömästä tilasta lähetetyn viestin funktiossa *CL_CheckForResend()*. Muodostettava komento on ”connect” parametrinaan pitkä merkkijono, joka sisältää muun muassa protokollan versionumeron, palvelimen haastenumeron, protokollaversion, qport-numeron sekä muita pelaajakohtaisia asetuksia kuten pelaajan nimimerkin. Muista yhteydenmuodostusprosessin aikana lähetettävistä viesteistä poiketen tästä merkkijonosta muodostetaan Huffmanin koodi. Koodin muodostamiseen käytetään poikkeuksellisesti mukautuvaa Huffmanin koodausta, sillä tavallisesti käytettävä staattisiin koodisanoihin perustuva koodaus ei tässä tapauksessa tuota toivottua tulosta. Pitkiä merkkijonona sisältävät viestit ovat ioquake3:n verkkoliikenteessä harvinaisia, mikä on nähtävissä myös liitteessä 1 esitetyistä suhdeluvuista ja koodisanoista.

Vastaanotettu viesti rekisteröidään palvelimella funktiossa *SV_ConnectionlessPacket()*, minkä jälkeen ajetaan yhdistysfunktio *SV_DirectConnect()*. Vastaavasti kuin asiakasohjelmassa aiemmin, myös palvelimella paikallista haastenumeroa verrataan asiakkaalta vastaanotettuun palvelimen haastenumeroon. Mikäli haastenumerot ja asiakkaan IP-osoite täsmäävät palvelimen tietojen kanssa, aloitetaan yhteydelliseen tilaan siirtyminen.

Asiakkaan lähettämä ”connect”-komento aiheuttaa palvelimella joukon erilaisia toimenpiteitä, jotka liittyvät niin sanotun verkkokanavan alustukseen. Quake-sarjan

protokollissa verkkokanavan avaus merkitsee yhteydelliseen tilaan siirtymistä ja näin ollen yhteydettömästä tilasta tapahtuvan kommunikaation lopettamista. Keskeisimmät *SV_DirectConnect()*-funktion toimenpiteet on kuvattu koodikatkelmassa 4.

```
1  void SV_DirectConnect( netadr_t from ) {
2  {
3      ...
4      // we got a newcl, so reset the reliableSequence and reliableAcknowledge
5      reliableAcknowledge = 0;
6      reliableSequence = 0;
7      ...
8      Netchan_Setup(NS_SERVER, &newcl->netchan, from, qport, challenge, qfalse);
9      // init the netchan queue
10     netchan_end_queue = &newcl->netchan_start_queue;
11     // send the connect packet to the client
12     NET_OutOfBandPrint(NS_SERVER, from, "connectResponse %d", challenge);
13     newcl->state = CS_CONNECTED;
14     newcl->lastSnapshotTime = 0;
15     newcl->lastPacketTime = svs.time;
16     newcl>lastConnectTime = svs.time;
17     ...
18     }
19 }
```

Koodikatkelma 4. Funktio sv_client.c : SV_DirectConnect() [The ioquake Group, 2014].

Kuten koodikatkelmasta 7 nähdään, ensimmänen oleellinen toimenpide on järjestysnumeroiden *reliableAcknowledge* ja *reliableSequence* nollaaminen. Kuten muuttujien nimistä on pääteltävissä, nämä järjestysnumerot liittyvät kiinteästi protokollan tapaan toteuttaa valikoiva luotettavuus. Luotettavuuden toteutusta tarkastellaan tarkemmin kohdassa 6.2.

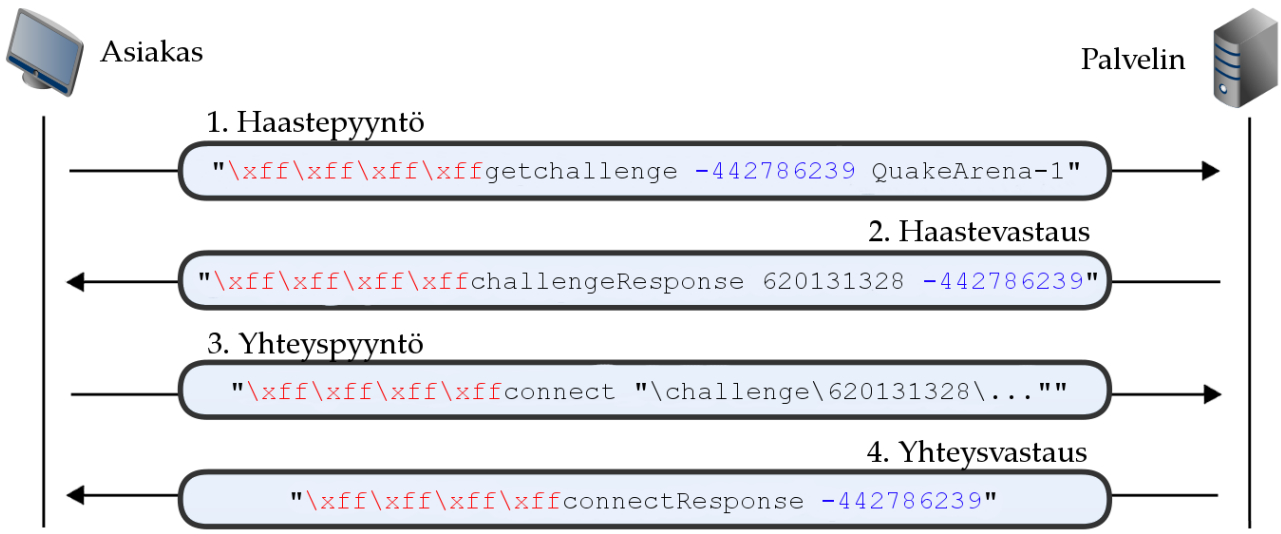
Järjestysnumeroiden nollauksen jälkeen ajetaan varsinainen verkkokanavan alustusfunktio *Netchan_Setup()*. Tämä funktio luo uudelle rekisteröidylle asiakkaalle oman verkkokanavatietueen, johon säilötään kaikki tarvittava tieto asiakkaasta protokollan toiminnallisuuden mahdollistamiseksi. Verkkokanavatietueen rakennetta esiteltiin tarkemmin kohdassa 5.3. Palvelimen verkkokanavatietueeseen tallennettava haastenumero on nimenomaan palvelimen aiemmin muodostama haastenumero.

Koodikatkelmaa 7 tarkasteltaessa huomataan, että verkkokanavan alustuksen jälkeen palvelin kutsuu *Netchan_OutOfBandPrint()*-funktiota lähettääkseen asiakkaalle yhteydettömästä tilasta komennon ”connectResponse”. Asiakasohjelma vastaanottaa palvelimen komennon tuttuun tapaan funktiolla *CL_ConnectionlessPacket()*. ”connectResponse”-komennon suorittaminen asiakasohjelmassa aiheuttaa verkkokanavan

alustuksen samaan tapaan kuin koodikatkelmassa 4 kuvatussa palvelimen tapauksessa. Päinvastoin kuin palvelimella, asiakkaan verkkokanavatietueeseen tallennetaan asiakkaan aiemmin muodostama haastenumero. Verkkokanavan alustuksen lisäksi verkkoyhteyden tilaksi asetetaan CA_CONNECTED, mikä määrittää verkkoyhteyden yhteydelliseen tilaan ja verkkokanavan käyttövalmiiksi.

Verkkokanavan molemminpuolisen alustuksen jälkeen tiedon lähetys ja vastaanotto tapahtuu pääsääntöisesti aina yhteydellisessä tilassa. Yhteydellisessä tilassa kaikki lähetetyt verkkopaketit – kaistan ulkopuolisia paketteja lukuunottamatta – noudattavat kohdassa 5.2. määriteltyä ioquake3-verkkopaketin rakennetta.

Aiemmissa luvuissa käsiteltyä yhteyden muodostamisprosessia on hyvä tarkastella vielä kokonaisuutena esimerkin avulla. Kuvassa 11 on esitetty kaikki yhteyden muodostamiseksi lähetettävät paketit esimerkkisisältöineen. Yhteyspyyntöviestin ”connect”-komennon parametrina lähetettävä merkkijono on esitetty selkokielisenä todellisuudessa lähetettävän Huffmanin koodin sijaan.Viestien eri osat on värikoodattu sisällön selkeyttämiseksi.



Kuva 11. Yhteydenmuodostusprosessin aikana muodostettavat verkkopaketit.

Kuvan 11 esimerkissä esiintyvistä numeroista -442786239 on asiakkaan muodostama haastenumero, 620131328 palvelimen haastenumero ja 70 protokollan versionumero. Kuten edellisessä alakohdassa mainittiin, haastepyynnön lähetyksen jälkeen asiakkaan verkkoyhteyden tilaksi asetetaan CA_CHALLENGING, yhteyspyynnön lähetyksen jälkeen CA_CONNECTING ja lopulta yhteysvastauksen suorittamisen jälkeen CA_CONNECTED. Verkkoprotokollan kulloinenkin tila vaikuttaa oleellisesti verkkokoodin toimintaan eri

tilanteissa. Mikäli asiakas vastaanottaa esimerkiksi haastevastauksen, kun verkkoyhteyden tilaksi on jo vaihdettu CA_CONNECTING, vastaanotettu paketti jätetään huomiotta.

Kuten kuvasta 11 nähdään, lähetettävien pakettien määrä yhteyden muodostamiseksi ei ole suuri, eikä prosessi ole muutenkaan lähetettävän tiedon kannalta monimutkainen. Verkkoprotokollan tapa siirtyä yhteydettömästä tilasta yhteydelliseen on kuitenkin oleellinen osa yhteydellisen sovelluskerroksen protokollan toteutusta. Periaatteessa kommunikointi yhteydellisessä tilassa voitaisiin aloittaa ensimmäisestä paketista lähtien, mutta tällöin jokaiselle mahdolliselle asiakkaalle täytyisi heti luoda oma verkkokanavatietue. Lienee selvää, että epätoivottujen verkkopakettien aiheuttamien vaikutusten minimoiminen alkuunsa on käytännöllisempi lähestysmistapa.

Ioquake3:n haaste-vaste -menetelmä antanee myös riittävän suojan sattumanvaraisten hyökkäyksien estoon, sillä sallittujen yhteydettömästä tilasta lähetettyjen komentojen joukko on erittäin rajallinen. Yhteydenmuodostuspakettien lisäksi ioquake3 sallii yhteydettömässä tilassa myös palvelinten etsintään liittyvät merkkijonopohjaiset viestit. Tuntemattomat asiakkaat eivät näillä viesteillä voi merkittävästi vaikuttaa käynnissä olevan pelin tapahtumiin. Quake 3:n palvelinhakutekniikkaa on käsitellyt yksityiskohtaisesti Armitage [2007].

5.4.2. Verkkokanavan funktioista

Ioquake3:ssa asiakas ja palvelin käyttävät tiedon lähetykseen ja vastaanottoon pääosin samoja funktioita, joskin yhteisten funktioiden kutsumiseen johtava funktioketju voi hieman vaihdella. Tiedon vastaanottamiseen liittyvässä funktioketjussa oleellisimpia funktioita ovat *Netchan_Process()* ja *NET_GetPacket()*. Vastaavat tiedon lähettämiseen käytettävät funktiot ovat *Netchan_Transmit()* ja *Sys_SendPacket()*. Lisäksi tietoa lähettäessä osana funktioketjua on myös *NET_SendPacket()*, joka vastaa lähinnä vuonohjauksesta. Yleensä verkkokanavaa vaativat funktiot alkavat etuliitteellä ”Netchan”, kun taas esimerkiksi pelkkä ”NET”-etuliite viittaa alemman tason funktioon. Verkkokanavaan viittaavat tiedonsiirtofunktiot ovat osa funktioketjua ainoastaan, mikäli verkkokanava on jo avattu.

NET_GetPacket()-funktio on vastaanotettujen verkkopakettien käsittelyfunktio, joka käyttää kuljetuskerroksen UDP-protokollaa uusien pakettien vastaanottamiseen. Näin ollen kyseistä funktiota voidaankin pitää rajapintana verkkoprotokollapinon sovellus- ja

kuljetuskerroksen välillä. *NET_GetPacket()* kutsuu kuljetuskerroksen UDP-protokollaa käyttöjärjestelmän kirjastojen funktiolla *recvfrom()*.

Mikäli *NET_GetPacket()* vastaanottaa kelvollisen verkkopaketin, saatu tieto lähetetään eteenpäin joko funktiolle *CL_PacketEvent()* tai *SV_PacketEvent()* riippuen onko käynnissä palvelin- vai asiakasohjelma. Yhteydellisessä tilassa *PacketEvent()*-funktiot tekevät muutaman virheentarkistuksen ja lähettävät verkkopaketin edelleen funktiolle *Netchan_Process()*. *Netchan_Process()* vastaa verkkopaketin otsikon käsittelystä ja mahdollisten sirpaloitujen pakettien hallinnasta. Mikäli verkkopaketti on kelvollinen, paketin sisältö siirtyy tämän jälkeen suoritettavaksi verkkokoodin ulkopuolelle.

Verkkokanavaa pitkin tietoa lähetettäessä funktioketju alkaa, kun asiakas- tai palvelinohjelma lähettää etäkohteessa suoritettavan komennon verkkoprotokollan funktiolle *Netchan_Transmit()*. Vastaavasti kuin edellisessä kappaleessa mainittu tiedon vastaanottoon tarkoitettu funktio *Netchan_Process()*, *Netchan_Transmit()* vastaa paketin otsikon kirjoittamisesta sekä mahdollisesta paketin sirpaloinnin hallinnasta. Otsikon ja viestin sisältävä verkkopaketti lähetetään tämän jälkeen edelleen funktiolle *NET_SendPacket()*. Tämä funktio joko asettaa lähettävän paketin jonoon tai lähettää sen suoraan eteenpäin funktiolle *Sys_SendPacket()*, joka toimii *NET_GetPacket()*-funktion tavoin rajapintana kuljetus- ja sovelluskerroksen välissä. Paketin lähettämiseen käytetty UDP-protokollan funktio on *sendto()*.

5.4.3. Luotettavuuden hallinta

Ioquake3-protokollan oleellisimpia ominaisuuksia UDP:hen verrattuna on protokollan valikoiva luotettavuus. Valikoivan luotettavuuden ansiosta verkkoliikennöinnissä voidaan hyödyntää sekä luotettavan että epäluotettavan protokollan hyviä puolia. Kuten aiemmin on todettu, suurin osa ioquake3:n verkkoliikenteestä on epäluotettavaa. On kuitenkin joitakin tapauksia, jolloin viestin saapuminen perille on välttämätöntä joko käyttökokemuksen tai ohjelman oikeanlaisen toiminnan kannalta.

Palvelimelta luotettavina lähetettäviä komentoja ovat muun muassa pelikartan vaihto ja pelaajien lähettämien keskusteluviestien tulostus. Niin ikään asiakkaan lähettämät palvelupyynnöt, kuten pyyntö ladata tiettyjä puuttuvia pelitiedostoja palvelimelta, lähetetään luotettavasti. Esimerkiksi kartanvaihdon yhteydessä lähetettävän käskyn katoaminen johtaisi luultavasti tilanteeseen, jossa pelaaja jäisi jumiin vanhaan pelikarttaan. Tällaisen tilanteen seuraukset ohjelman toiminnassa jatkossa olisivat arvaamattomat.

Kuten aiemmin kohdassa 5.2. mainittiin, verkkopakettien rakenteessa on määritelty oma kenttä luotettaville viesteille ja viimeksi vastaanotetun luotettavan viestin järjestysnumerolle. Nämä kentät ovat avainasemassa valikoivan luotettavuuden toteutuksessa. Koska verkkopakettien mukana lähetetään erikseen tieto vastaanotetun verkkopaketin järjestysnumerosta ja vastaanotetun luotettavan viestin järjestysnumerosta, voidaan viestien lähetys toteuttaa valikoivasti.

Valikoivalla luotettavuudella on taipumus aiheuttaa vaikeasti havaittavia virhetilanteita, joissa luotettavat viestit viittaavat epäluotettavien viestien aikaisemmin muokkaamiin pelientiteetteihin [Hook, 2004]. Toinen ongelmatilanne syntyy päinvastaisessa tilanteessa, jossa epäluotettava viesti viittaa vielä saapumattoman luotettavan viestin muokkaamiin tietoihin. Näin saattaa käydä, koska UDP-protokolla ei takaa pakettien saapumista oikeassa järjestyksessä. Havainnollistetaan seuraavaksi, miten nämä ongelmat on ratkaistu ioquake3:ssa.

Ioquake3:n delta-pakkaus on oleellinen osa ensimmäisen ongelmatilanteen ratkaisua. Koska uudet tilannevedokset muodostetaan aina asiakkaan viimeksi kuittaaman tilannevedoksen pohjalta (ks. alakohta 5.5.2.), eivät uudet tilannevedokset koskaan viittaa olemattomaan tietoon. Delta-pakkauksen toimintamekanismi edellyttää palvelimelta tietoa asiakkaan kuittaamista pelitiloista, joten luotettavatkin viestit voidaan muodostaa asiakkaan tunnetun pelitilan perusteella. Epäluotettavien viestien katoaminen matkalla ei haittaa, koska kuittaamattomia tilannevedoksia ei käytetä pohjana muodostettavalle epäluotettavalle tai luotettavalle viestille.

Toinen ongelmatilanne, jossa vastaanotettu epäluotettava viesti viittaa vielä saapumattoman luotettavan viestin muokkaamiin tietoihin, on ratkaistu ioquake3:ssa varsin yksinkertaisesti: luotettava viesti lisätään jokaiseen lähtevään verkkopakettiin, kunnes viesti on kuitattu vastaanotetuksi. Jokainen muodostettu verkkopaketti sisältää siis saman luotettavan viestin, vaikka epäluotettava osio vaihtuisikin tilannevedoksen päivityksen myötä. Luotettavan viestin kuittausta ei varsinaisesti jäädä odottamaan, vaan protokolla toimii normaaliin tahtiin lisäten jokaiseen lähtevään pakettiin vielä kuittaamattomat luotettavat viestit. Luotettavat viestit suoritetaan aina ennen tilannevedosten lukua, mikä varmistaa mahdollisten luotettavien muutosten olemassaolon.

Toisin kuin esimerkiksi QuakeWorld [Id Software, 1999] ja Quake 2 [Id Software, 2001], ioquake3 tukee useiden luotettavien viestien samanaikaista lähettämistä. Koska jokainen

luotettava viesti lisätään osaksi lähteviä verkkopaketteja kuittauksen vastaanottoon saakka, voidaan kaikkien viestien vastaanotosta varmistua järjestysnumeron osoittaessa viimeisimpään luotettavaan viestiin. UDP-protokolla takaa pakettien saapumisen joko kokonaisena tai ei ollenkaan, minkä ansiosta kaikki paketin luotettavat viestit on varmasti vastaanotettu kuittauksen saapuessa.

Ioquake3:ssa luotettavat viestit ovat tavallisista epäluotettavista viesteistä poiketen yleensä merkkijonoja. Viestit ovat samantyyppisiä kuin yhteydenmuodostusprosessissa käytettävät merkkijonoviestit – esimerkiksi yhteyden katkaisu tapahtuu luotettavalla viestillä ”disconnect”. Viestit suoritetaan vastaanottajalla omissa funktioissaan lähetetyn komennon perusteella. Funktioiden tekninen toteutus on varsin suoraviivainen, joten yksityiskohtaiselle tarkastelulle ei liene tässä tarvetta. Komentojen suorittamistapa on luonnollisesti hyvin pelikohtaista, joten oleellista tässä onkin ymmärtää valikoivan luotettavuuden korkeamman tason toimintaperiaate ja sen merkitys pelin toiminnassa.

Luotettavien viestien puskuriin kirjoittamiseen käytetään asiakaspuolella cl_main.c-lähdekooditiedoston funktiota *CL_AddReliableCommand()*. Vastaava palvelimen funktio on sv_main.c-tiedoston *SV_AddServerCommand()*. Viestit kopioidaan puskurista lähtevään verkkopakettiin verkkokanavan funktiossa *Netchan_Transmit()*, ja vastaavasti kopioidaan vastaanotetusta paketista puskuriin funktiossa *Netchan_Process()*. Luotettavien viestien lähettäminen vaatii luonnollisesti avattua verkkokanavaa.

Joissakin tapauksissa luotettavat viestit saattavat olla niin pitkiä, että verkkopaketti on syytä sirpaloida useampaan osaan. Tarkastellaan seuraavaksi ioquake3:n toimintaa tällaisessa tilanteessa.

5.4.4. Verkkopakettien sirpalointi

Kuten kohdassa 5.2. mainittiin, ioquake3 määrittelee verkkopaketin maksimikooksi 1400 tavua. Quake 3:n lähettämien pakettien keskimääräinen koko on tähän verrattuna erittäin pieni [Pavlicic & Armitage, 2003; Pozzobon, 2002], ja näin isot verkkopaketit ovatkin erittäin harvinaisia. Poikkeustilanteen saattaa kuitenkin aiheuttaa esimerkiksi täyden tilanpäivityksen sisältävä verkkopaketti, joka lähetetään yleensä asiakkaan liittyessä peliin. Myös useat pitkät luotettavat viestit samassa verkkopaketissa saattavat aiheuttaa paketin koon kasvamisen poikkeuksellisen isoksi.

Viestien sirpaloinnin tarve tarkistetaan verkkokanavan normaalissa lähetysfunktiossa *Netchan_Transmit()*. Vaikka lähdekoodin kommenteissa mainitaan sirpaloinnin yhteydessä ainoastaan luotettavat viestit, ei viestin luotettavuutta tarkisteta erikseen. Käytännössä epäluotettavat viestit eivät kuitenkaan koskaan saavuta 1400 tavun rajaa, joten tarkistukselle ei ioquake3:n tapauksessa myöskään ole tarvetta. Mikäli *Netchan_Transmit()* havaitsee verkkopaketin koon ylittävän maksimirajan, lähetettävä viesti ohjataan funktiolle *Netchan_TransmitNextFragment()*. Funktion toimintaa on kuvattu koodikatkelmassa 5.

```
1  void Netchan_TransmitNextFragment( netchan_t *chan ) {
2     ...
3     outgoingSequence = chan->outgoingSequence | FRAGMENT_BIT;
4     MSG_WriteLong(&send, outgoingSequence);
5
6     // send the qport if we are a client
7     if ( chan->sock == NS_CLIENT ) {
8        MSG_WriteShort( &send, qport->integer );
9     }
10    ...
11    fragmentLength = FRAGMENT_SIZE;
12    if ( chan->unsentFragmentStart  + fragmentLength > chan->unsentLength ) {
13       fragmentLength = = chan->unsentLength - chan->unsentFragmentStart;
14    }
15
16    MSG_WriteShort( &send, chan->unsentFragmentStart );
17    MSG_WriteShort( &send, fragmentLength );
18    MSG_WriteData( &send, chan->unsentBuffer + chan->unsentFragmentStart,
19       fragmentLength );
20
21    // send the datagram
22    NET_SendPacket(chan->sock, send.cursize, send.data, chan->remoteAddress);
23    ...
24    chan->unsentFragmentStart += fragmentLength;
25    ...
26 }
```

Koodikatkelma 5. Funktio net_chan.c : Netchan_TransmitNextFragment() [The ioquake Group, 2014].

Kuten koodikatkelmasta 5 nähdään, sirpaloituun verkkopakettiin lisätään normaaliin tapaan lähtevän paketin järjestysnumero. Tämän lisäksi järjestysnumeroa vastaavan bittijonon merkitsevin bitti asetetaan poikkeuksellisesti päälle bittitason TAIlla. Bittitason operaation operandeina ovat verkkopaketin järjestysnumeroa vastaava kokonaisluku ja vakio FRAGMENT_BIT. Edellä mainitun vakion arvoksi on määritelty net_chan.c-tiedostossa (1<<31), joka vastaa käytännössä bittijonoa, jossa vain merkitsevin bitti on päällä. Näin ollen bittitason operaation tuloksena saadaan bittijono, joka vastaa merkitsevintä bittiä lukuunottamatta verkkopaketin järjestysnumeroa. Merkitsevimmän bitin käyttäminen sirpaloidun paketin tunnusmerkkinä ei aiheuta ongelmia, sillä

verkkopaketin järjestysnumero ei käytännössä koskaan kasva yli 32-bittisen kokonaisluvun maksimiarvon. Mikäli näin tapahtuisi, ohjelma kaatuisi luultavasti joka tapauksessa puskurin ylivuotovirheeseen.

Koodikatkelman 5 riveiltä 6-8 nähdään, että seuraavaksi verkkopakettiin kirjoitetaan normaaliin tapaan qport-numero, mikäli lähettäjä on asiakas. Riveillä 11-13 määritellään lähetettävän sirpaleen pituus. Oletuksena käytetään viestisirpaleen maksimikokoa eli vakiota FRAGMENT_SIZE, joka vastaa 1300 tavua. Voidaan olettaa, että ”puuttuvat” 100 tavua varataan varmuuden vuoksi otsikolle ja viestiosan järjestysnumeroille, sillä muuttujan fragmentLength arvo vastaa vain verkkopaketin viestiosion varsinaista sisältöosaa. Mikäli lähetettävästä viestistä jäljellä olevan osan pituus ei ylitä 1300 tavun rajaa, asetetaan sirpaleen pituudeksi jäljellä olevan osan pituus.

Seuraavaksi verkkopakettiin kirjoitetaan viestisirpaleen sijainti suhteessa viestin alkuun, sirpaleen pituus sekä itse viestin osa. Sirpale lähetetään vastaanottajalle funktiolla *NET_SendPacket()*, minkä jälkeen ähetettävän viestisirpaleen sijaintiin viittaavan muuttujan unsentFragmentStart arvoon summataan lähetetyn viestisirpaleen pituus. Mikäli lähetettävää tietoa jäi jäljelle, funktio ajetaan uudestaan pelisilmukan seuraavalla suorituskerralla. Huomionarvoista on, että kaikilla sirpaloidun verkkopaketin sirpaleilla on sama järjestysnumero.

Vastaanottajan päässä sirpaleiden yhteen kokoaminen perustuu sirpaleisiin kirjoitettuun tietoon viestisirpaleen pituudesta sekä sijainnista viestissä. Sirpaleet kirjoitetaan puskuriin funktiossa *Netchan_Process()*. Kun kaikki sirpaleet on vastaanotettu, kirjoitetaan sirpaleista koottu kokonainen viesti normaaliin viestipuskuriin, josta erinäiset vastaanotettuja viestejä suorittavat funktiot viestin lukevat.

Vastaanotettua sirpaloitua viestiä ei kuitata vastaanotetuksi ennen kuin kaikki sirpaleet ovat saapuneet, joten mahdolliset matkalla hukkuneet sirpaleet lähetetään luotettavan periaatteen mukaisesti uudelleen. Uudelleenlähetettäviä sirpaleita ei voida yksilöidä, koska jokaisella sirpaleella on sama verkkopaketin järjestysnumero. Näin ollen jonkin sirpaleen hukkuessa jokainen sirpale lähetetään uudestaan. Vastaanottajan jo lukemat sirpaleet säilyvät kuitenkin sirpalepuskurissa, ja uudelleen vastaanotetut sirpaleet jätetään huomiotta. Periaatteessa sirpaleiden vastaanotto perustuu yksinkertaisesti siihen, että vain oikeassa järjestyksessä saapuneet sirpaleet luetaan puskuriin ja loput jätetään huomiotta. Protokollan luotettavuus takaa, että puuttuvat sirpaleet saapuvat perille ennen pitkää.

5.5. Verkkoliikenteen minimointi

Kuten taulukossa 5 mainittiin, ioquake3 käyttää tietoliikenteen minimoimiseen kahta erilaista pakkausmenetelmää: Huffmanin koodausta ja delta-pakkausta. Ennen verkkopakettien pakkausta suoritetaan kiinnostuksenhallintaa, joka määrää, mitä tietoa pakattavaksi ylipäätään siirretään. Lisäksi verkkopakettien dynaaminen viestirakenne mahdollistaa verkkopaketin koon määrittämisen sen mukaan, mitä tietoa paketti kullakin lähetyskerralla sisältää. Tarkastellaan verkkoliikenteen minimointiin liittyviä ominaisuuksia niiden käytännön suoritusjärjestyksessä: ensimmäisenä kiinnostuksenhallinta, toisena delta-pakkaus ja viimeisenä Huffmanin koodaus.

5.5.1. Kiinnostuksenhallinta

Kuten Quake-sarjan aikaisemmat pelit [Id Software, 1999; 2001], myös Quake 3 [Id Software, 2005] tallentaa pelikartat BSP-muodossa ennaltalaskettujen solmukohtaisten PVS-tietojen kera [Sanglard, 2009]. Ioquake3 ei poikkea tässä tapauksessa toiminnallisuudeltaan alkuperäisestä Quake 3:sta. Koska PVS-tietojen laskeminen BSP-puun jokaiselle solmulle on nähtävä enemmän tietokonegrafiikkaan kuin verkkotekniikkaan liittyvänä ongelmana, tähän käytettävien algoritmien käsittely lienee perusteltua nyt sivuuttaa. Quake-sarjan käyttämiä BSP- ja PVS-algoritmeja on tutkittu ja havainnollistettu olemassaolevassa kirjallisuudessa hyvin kattavasti [Abrash, 1997a; 1997b; Sanglard, 2009; van Waveren, 2007].

Lyhyesti ioquake3:n PVS:n perusteella tapahtuva kiinnostuksenhallinta toimii tarkistamalla pelaajan sijainti BSP-puussa ja käyttämällä hyväksi sijaintia vastaavan solmun ennaltalaskettuja PVS-tietoja. PVS-tietojen perusteella poissuljettuja pelientiteettejä ei piirretä, eikä niitä myöskään lähetetä verkon yli.

PVS:ään perustuvan kiinnostuksenhallinnan lisäksi ioquake3 tukee pelientiteeteille määriteltäviä poikkeuksia verkkoteknisen kiinnostuksenhallinnan suhteen. Nämä poikkeukset on listattu taulukossa 9. Taulukon 9 vakioiden lisäksi ioquake3:ssa on muitakin SVF-alkuisia vakioita, mutta nämä eivät liity verkkotekniseen kiinnostuksenhallintaan.

Poikkeukset on määritelty bitteinä, joiden asettaminen päälle entiteetin entityShared_t-tyyppisen tietueen muuttujassa svFlags laukaisee halutun toiminnallisuuden. Jokainen taulukon 9 vakiota vastaava kokonaisluku on binääriesitykseltään bittijono, jossa on vain yksi päällä oleva bitti. Tämän ansiosta yksi kokonaisluku riittää yhdestä kaikkien poikkeuksien määrittämiseen jollekin pelientiteetille.

Poikkeuksen olemassaolo tarkistetaan bittitason JAlla käyttäen operandeina pelientiteetin svFlags-muuttujaa ja poikkeukselle määritettyä taulukon 9 vakiota.

Vakio	*Kokonaisluku*	*Vaikutus*
SVF_NOCLIENT	1	Entiteettiä ei koskaan lähetetä verkon yli yhdellekään asiakkaalle.
SVF_CLIENTMASK	2	Entiteetti lähetetään vain tietylle ryhmälle asiakkaita.
SVF_BROADCAST	32	Entiteetti lähetetään aina jokaiselle asiakkaalle.
SVF_PORTAL	64	PVS:n laajennuspiste, joka aiheuttaa toisen BSP-solmun PVS:n yhdistämisen nykyiseen PVS:ään ja tilannevedokseen.
SVF_SINGLECLIENT	256	Entiteetti lähetetään vain tietylle asiakkaalle.
SVF_NOTSINGLECLIENT	2048	Entiteetti lähetetään kaikille asiakkaille yhtä poikkeusta lukuunottamatta.

Taulukko 9: Verkkoteknisen kiinnostuksenhallinnan poikkeukset. Johdettu lähdekoodi-tiedostoista g_public.h ja sv_snapshot.c [The ioquake Group, 2014].

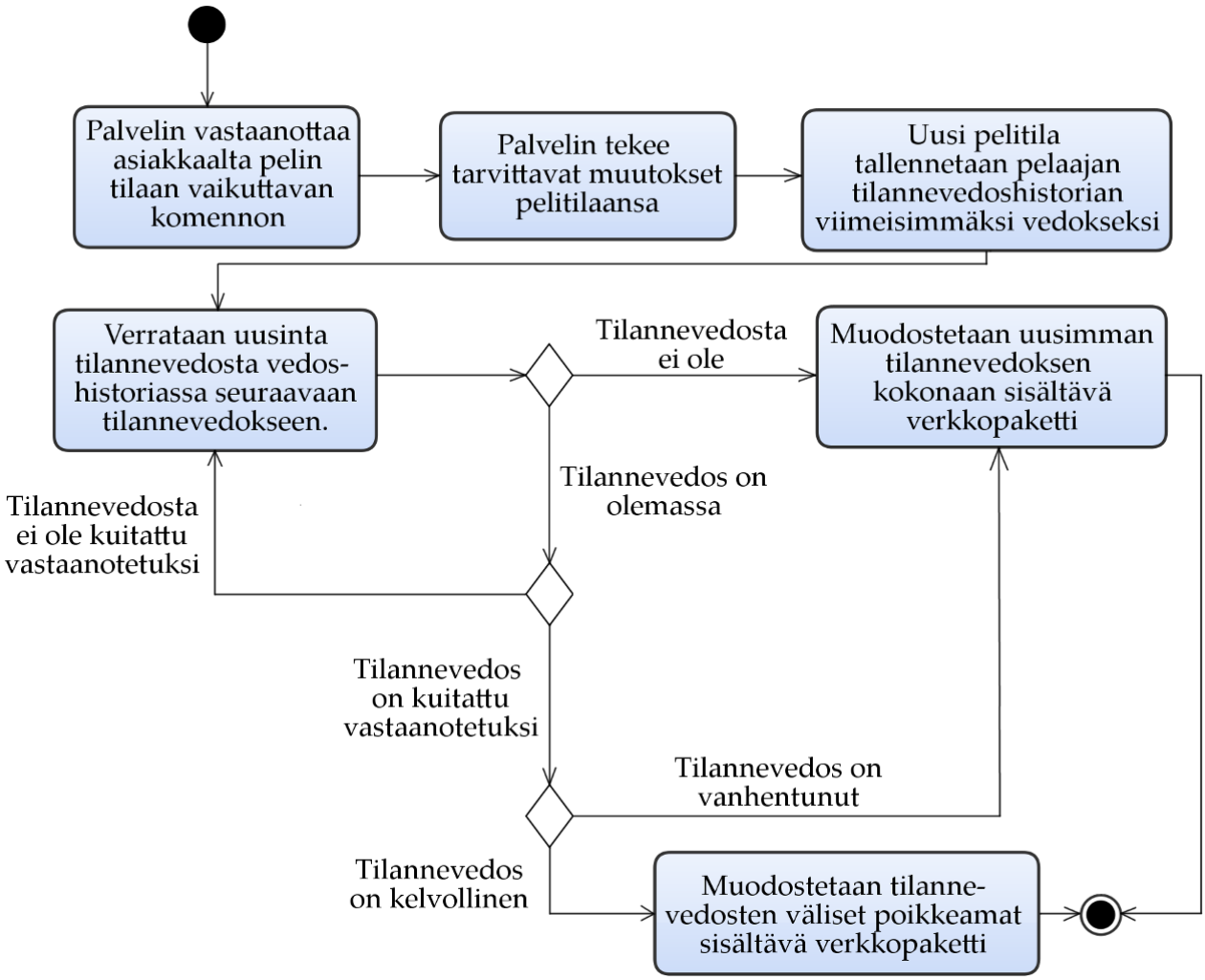
Taulukon 9 kaltaiset vakiot ovat oiva tapa määritellä poikkeuksia pelientiteeteille, joiden lähettäminen kaikille asiakkaille halutaan syystä tai toisesta estää. Koska poikkeusten määrittely perustuu bittijonoihin ja testaus bittitason operaatioihin, voidaan toteutusta pitää myös varsin tehokkaana. BSP-puuhun ja PVS:ään perustuvan kiinnostuksenhallinnan kanssa tämä ominaisuus muodostaa melko tehokkaan tavan rajoittaa turhaa verkkoliikennettä jo ennen verkkopaketin pakkausta.

5.5.2. Delta-pakkaus

Kuten alakohdassa 3.2.1. mainittiin, delta-pakkauksella tarkoitetaan tekniikkaa, joka perustuu vain muuttuneen tiedon lähettämiseen täyden pelitilan päivityksen sijasta. Ioquake3:ssa kyseinen tekniikka on toteutettu teknisesti mielenkiintoisesti ja tehokkaasti tilannevedoksia hyödyntäen.

Alkuperäisen Quake 3:n tapaan myös ioquake3-palvelin muodostaa jatkuvasti uusia pelaajakohtaisia tilannevedoksia, joihin sisällytettävä tieto perustuu pelaajan kulloiseenkin sijaintiin ja näkökenttään pelimaailmassa. Kuten lähdekooditiedoston cg_public.h [The ioquake Group, 2014] tilannevedostietuetta snapshot_t tarkasteltaessa selviää, tilannevedokset sisältävät odotetusti muun muassa pelaajan omaa tilaa kuvaavan tietueen sekä tietuetaulukon muiden oleellisten pelientiteettien tilasta. Quake 3:n pelientiteettitietueita ovat käsitelleet yksityiskohtaisesti Stefyn ja muut [2011].

Kiinnostavinta ioquake3:n delta-pakkauksessa on varsinainen tilannevedosten vertailuprosessi, sillä tekniikka lienee tältä osin sovellettevissa myös muihin reaaliaikaisiin internet-peleihin. Ioquake3:n delta-pakkauksen toimintaperiaate yleisellä tasolla on esitetty kuvassa 12. Tekniikka perustuu viimeisimmän vastaanotetuksi kuitatun tilannevedoksen vertaamiseen palvelimen absoluuttiseen pelitilaan.



Kuva 12: Delta-pakkaus ioquake3:ssa [The ioquake Group, 2014; Sanglard, 2012].

Kuten kuvasta 12 nähdään, ainoastaan vastaanotetuksi kuitattuja tilannevedoksia voidaan käyttää uuden tilannevedoksen muodostamiseen. Kuittaamatonta tilannevedosta ei siis voida käyttää vertailuun, vaikka se olisikin tallennettu palvelimen ylläpitämään tilannevedoshistoriaan. Verkkopakettien koko kasvaa näin ollen samassa suhteessa kadonneiden tilannevedosten lukumäärän kanssa, sillä oletettavasti muuttuneiden arvojen määrä on aina suurempi vanhempaan tilannevedokseen verrattuna.

Tilannevedosten vastaanottamisen vahvistaminen on välttämätöntä, jotta laskettu muutos tuottaa toivotun lopputuloksen asiakkaan pelitilassa. Mikäli kuitattuja tai muuten käyttökelpoisia tilannevedoksia ei ole, lähetetään asiakkaalle täysi tilanpäivitys. Tällainen verkkopaketti on luonnollisesti kooltaan huomattavasti suurempi kuin tavalliset delta-pakatut verkkopaketit, eikä täyttä tilanpäivitystä lähetetäkään tavallisesti muulloin kuin asiakkaan liittyessä peliin tai pelikartan vaihtuessa.

Tarkastellaan seuraavaksi tarkemmin ioquake3:n käytännön tapaa vertailla pelientiteettien tilatietueita ja laskea tapahtunut muutos tilannevedoksen muodostamiseksi. Tämä voidaan aloittaa tarkastelemalla msg.c-lähdekooditiedoston [The ioquake Group, 2014] tietuetta entityStateFields, jota käytetään entityState_t-tyyppisten pelientiteettien tilatietueiden vertailuun.

```
1   typedef struct {
2     char  *name;
3     int         offset;
4     int         bits;       // 0 = float
5   } netField_t;
6
7   // using the stringizing operator to save typing...
8   #define NETF(x) #x,(size_t)&((entityState_t*)0)->x
9
10  netField_t entityStateFields[] =
11  {
12    { NETF(pos.trTime), 32 },
13    { NETF(pos.trBase[0]), 0 },
14    ...
15    { NETF(event), 10 },
16    { NETF(modelindex), 8 },
17    ...
18  }
```

Koodikatkelma 6. Tietue msg.c : entityStateFields [The ioquake Group, 2014].

Koodikatkelman 6 netField_t-muotoinen entityStateFields-tietue sisältää pelientiteetin tilatietueen jokaisen kentän nimen, sijainnin ja koon, mikä mahdollistaa kenttien vertailun ”sokeasti” kokonaislukuina. Rivillä 8 esitetty direktiivi on avainasemassa tämän vertailuun

käytettävän tietueen muodostamisessa, joten kyseisen rivin selventäminen lienee paikallaan.

Rivillä 8 määritellyn direktiivin suorittamista toimenpiteistä ensimmäinen on #x, jossa operaattori # muuntaa parametrina saadun entityState_t-muotoisen pelientiteetin tilatietueen kentän nimen lainausmerkeillä rajatuksi merkkijonoksi. Käytännössä tällä määritellään entityStateFields-tietueen alkion ensimmäinen kenttä eli merkkijono name.

Seuraava toimenpide asettaa entityStateFields-tietueen toisen kentän eli kentän offset arvoksi parametrina saadun tilatietueen kentän suhteellisen sijainnin. Saatu tulos tallennetaan entityStateFields-tietueeseen kokonaislukuna, joka vastaa kentän etäisyyttä tavuina tietueen alusta. Kuten riviltä 8 on nähtävissä, sijainti saadaan ensin muuntamalla kokonaisluku 0 osoittimeksi entityState_t-tyyppiseen tilatietueeseen. Kokonaisluvun 0 käyttäminen ei tässä tapauksessa ole sattumaa, sillä 0:n muuntaminen osoittimeksi asettaa osoittimen kohteen muistiosoitteen alkavaksi arvosta 0. Osoitin asetetaan seuraavaksi viittaamaan parametrina saatuun tietuekenttään, minkä jälkeen tämän tietuekentän muistiosoite muunnetaan size_t-tyyppiseksi kokonaisluvuksi. Koska viitattavan tietueen muistiosoite alkoi 0:sta, saadaan haluttuun kenttään viittaamalla kentän etäisyys tavuina tietueen alusta. Olemattomaan muistiosoitteeseen viittaaminen ei tässä tapauksessa aiheuta ongelmia, koska kyseisessä osoitteessa näennäisesti sijaitsevaan tietueeseen ei koskaan yritetä päästä käsiksi.

Sijaintitietojen lisäämisen jälkeen taulukkoon lisätään kunkin kentän koko bitteinä koodikatkelman 6 riveillä 12-16 nähtyyn tavanomaiseen tapaan. Näiden toimenpiteiden mahdollistama tilannevedosten sisältämien pelientiteettien tilatietueiden vertailu ja sen tuloksen perusteella tehtävä verkkopaketin kirjoitus on esitetty koodikatkelmassa 7. Koodikatkelmassa kuvattu toiminnallisuus toistetaan jokaiselle tilatietueen kentälle.

```
1   void MSG_WriteDeltaEntity ( msg_t *msg, struct entityState_s *from, struct
2         entityState_s *to, qboolean force ) {
3       ...
4       fromF = (int *)( (byte *)from + field->offset );
5       toF = (int *)( (byte *)to + field->offset );
6
7       if ( *fromF == *toF ) {
8          MSG_WriteBits( msg, 0, 1 );      // no change
9          ontinue;
10       }
11      MSG_WriteBits( msg, 1, 1 );  // changed
12
13      if ( field->bits == 0 ) {
14         // float
15         ...
16      } else {
17         if (*toF == 0) {
18            MSG_WriteBits( msg, 0, 1 );
19         } else {
20            MSG_WriteBits( msg, 1, 1 );
21            // integer
22            MSG_WriteBits( msg, *toF, field->bits );
23         }
24      }
25     ...
26    }
```

Koodikatkelma 7. Funktio msg.c : MSG_WriteDeltaEntity() [The ioquake Group, 2014].

Kuten koodikatkelman 7 riveiltä 4 ja 5 nähdään, kenttien vertailu tapahtuu muutamien tyyppimuunnosten kautta. Käsiteltävät pelientiteettien tilatietueet muunnetaan ensin byte-tyyppisiksi osoittimiksi, minkä jälkeen osoitin kohdistetaan oikeaan kenttään lisäämällä osoittimeen koodikatkelmassa 6 määritelty tavumäärä. Tietue field on siis tyyppiä netField_t ja vastaa sisällöltään tietuetta entityStateFields, joten arvo offset viittaa kentän etäisyyteen tilatietueen alusta. Osoittimen sijainti muuttuu odotetulla tavalla, koska osoittimen tietotyyppinä käytetty byte on nimensä mukaisesti yksitavuinen tietotyyppi. Lopulta osoittimet muunnetaan vielä int-tyyppisiksi, jolloin tietueiden sisältöä voidaan verrata neljä tavua kerralla.

Kuten koodikatkelmasta 7 nähdään, itse vertailu tapahtuu yksinkertaisesti vertailemalla niitä neljätavuisia bittijonoja, joihin osoittimet kullakin silmukan suorituskerralla viittaavat. Tilatietueiden kenttien oikealla tietotyypillä ei ole merkitystä, mikä tekee tästä ratkaisusta erityisen mielenkiintoisen ja oletettavasti myös tehokkaan.

Mikäli vuorossa olevat bittijonot vastaavat toisiaan, kirjoitetaan yksi pois päältä oleva bitti funktiota *MSG_WriteBits()* käyttäen. Käytännössä tässä toteutuu myös verkkopaketin

dynaaminen rakenne, sillä pois päältä oleva bitti korvaa tavallisesti samassa kohdassa tilannevedospaketissa olevan kentän. Vastaanottaja osaa siirtyä pois päältä olevan bitin perusteella seuraavaan kenttään, joten muuttumatonta kenttää ei tarvitse kirjoittaa pakettiin turhaan. Tämä on juuri sellainen tarpeellinen toimenpide, johon viitattiin delta-pakkauksen esittelyn yhteydessä alakohdan 3.2.1. lopussa.

Mikäli vuorossa olevat bittijonot taas poikkeavat toisistaan, kirjoitetaan ensin yksi päällä oleva bitti. Tämän jälkeen tutkitaan, onko vuorossa oleva tilatietueen kenttä liukuluku vai kokonaisluku ja onko luvun arvo 0. Mikäli kentän arvo on kokonaisluku 0, riittää tiedon lähettämiseksi yhden pois päältä olevan bitin lisääminen aiemmin lisätyn päällä olevan bitin jatkoksi. Mikäli kyseessä on muu kokonaisluku, kirjoitetaan vuorossa olevan tilatietueen kentän pituutta vastaava määrä bittejä osoittimen viittaamasta sijainnista. Liukulukujen tapauksessa tarkistetaan, riittääkö liukuluvun kuvaamiseen vähemmän bittejä käyttävä kokonaislukutyyppiä soveltava esitystapa, vai täytyykö liukuluku lähettää kokonaisuudessaan 32-bittisenä. Bittien kirjoittamisen jälkeen jatketaan seuraavaan tilatietueen kenttään, kunnes kaikki kentät on käsitelty.

On syytä huomioida, että esimerkiksi pelaajan omaa pelihahmoa vastaavan entiteetin tilatietue on nyt käsitellyistä tietueista poiketen tyyppiä playerState_t, mutta delta-koodaus toimii tästä huolimatta samalla periaatteella. Pelaajan oman hahmon tapauksessa koodikatkelman 6 tietuetta entityStateFields vastaa tietue playerStateFields ja direktiiviä NETF direktiivi PSF. Kuten entityStateFields ja NETF, myös playerStateFields ja PSF määritellään lähdekooditiedostossa msg.c. Eri tietuetta käytetään, koska pelaajan oman pelihahmon tilasta voidaan lähettää erilaista tietoa kuin muista pelientiteeteistä.

Tilannevedosta vastaanottaessa ja luettaessa ioquake3 toimii täsmälleen päinvastaisesti nyt kuvattuun prosessiin verrattuna, joten vastaanottoprosessin tarkka kuvaaminen ei liene tarpeen. Seuraavaksi onkin luontevaa siirtyä käsittelemään ioquake3:n Huffmanin koodausta, sillä tässä alakohdassa käsitelty delta-pakkausfunktiokin käyttää viestien kirjoittamiseen kiinteästi Huffmanin koodaukseen liittyvää funktiota *MSG_WriteBits()*. Ennen yksityiskohtaiseen funktioiden tarkasteluun siirtymistä on kuitenkin syytä kuvata ioquake3:n tapaa toteuttaa Huffmanin koodaus ylemmällä tasolla.

5.5.3. Huffmanin koodaus

Ioquake3:n lähdekooditiedoston huffman.c kommenteissa mainitaan, että protokollan käyttämä koodausperiaate perustuu mukautuvaan Huffmanin koodaukseen. Lähdekoodia tutkimalla voidaan kuitenkin havaita, että aidosti mukautuvaa Huffmanin koodausta käytetään ainoastaan verkkoyhteyden muodostamisprosessissa (ks. kohta 6.1.). Muissa tapauksissa Huffmanin koodi noudattaa tavallisen Huffmanin koodauksen tapaan ennalta sovittuja koodisanoja, joita ei lähetetä muodostetun koodin mukana. Lähdekoodissa oleva kommentti mukautuvasta Huffmanin koodauksesta viitannee lähinnä ioquake3:n tapaan muodostaa staattinen Huffmanin puu, joka määrittää käytetyt koodisanat. Tältä osin ohjelman toimintatapa on täysin sama kuin alakohdassa 3.2.2. esitetyssä Sayoodin [2012] mukautuvassa Huffmanin koodauksessa.

Ioquake3:n Huffmanin koodaus toimii tavuittain, joten koodien aakkostoksi voidaan asettaa kahdeksanbittisen kokonaislukutyypin määrittelyjoukko [0, 255]. Koodauksessa käytettävä staattinen Huffmanin puu sisältää näin ollen koodisanan jokaiselle yksitavuiselle bittijonolle. Muodostettava Huffmanin puu perustuu oletettavasti Quake 3:n kehittäjien keräämiin tilastoihin eri kokonaislukujen esiintymistiheyksistä. Koodisanoja vastaavat kokonaisluvut, niiden esiintymistiheydet, suhdeluvut ja mahdolliset tulostettavat arvot on koottu liitteeseen 1. Liitteen 1 Huffmanin puun viimeinen solmu on puun muodostamiseen käytettävä NYT-solmu, joka ei kohdan 6.1. poikkeusta lukuun ottamatta esiinny koskaan osana valmista Huffmanin koodia.

Oleellisimpia Huffmanin koodin muodostamiseen liittyviä ioquake3:n funktioita ovat msg.c-lähdekooditiedoston *MSG_WriteBits()* sekä huffman.c-tiedoston *Huff_putBit(), Huff_offsetTransmit(), send()* ja *add_bit()*. Edellä mainituista funktioista ensimmäistä kutsutaan aina, kun lähetettävään verkkopakettiin kirjoitetaan tietoa. Eripituisia bittijonoja kirjoittaessa saatetaan käyttää välillisesti esimerkiksi funktiota *MSG_WriteByte()* tai *MSG_WriteLong()*, mutta lopulta kirjoittaminen suoritetaan aina funktion *MSG_WriteBits()* avulla. Koodien purkamiseen käytetään vastaavasti nimettyjä funktioita *MSG_ReadBits()*, *Huff_getBit()*, *Huff_offsetReceive()* ja *get_bit()*.

Huffmanin koodaus staattiseen puuhun perustuen suoritetaan automaattisesti, mikäli kyseessä ei ole kaistan ulkopuolinen paketti. Kaistan ulkopuolisista paketeista ainoastaan yhdestä muodostetaan Huffmanin koodi, joka sekin muodostetaan poikkeuksellisesti mukautuvalla Huffmanin koodauksella. Tähän käytetään funktioita *Huffman_Compress()* ja

Huffman_Decompress(). Staattiseen Huffmanin puuhun perustuvassa koodauksessa käytettävää funktiota *MSG_WriteBits()* on havainnollistettu koodikatkelmassa 8.

```
1  void MSG_WriteBits( msg_t *msg, int value, int bits ) {
2     ...
3     value &= (0xffffffff>>(32-bits));
4     if (bits&7) {
5        int nbits;
6        nbits = bits&7;
7        for(i=0;i<nbits;i++) {
8           Huff_putBit ((value&1), msg->data, &msg->bit);
9           value = (value>>1);
10       }
11       bits = bits - nbits;
12    }
13    if (bits) {
14       for(i=0;i<bits;i+=8) {
15          Huff_offsetTransmit (&msgHuff.compressor, (value&0xff), msg->data,
16             &msg->bit);
17          value = (value>>8);
18       }
19    }
20    ...
21 }
```

Koodikatkelma 8. Funktio msg.c : MSG_WriteBits() [The ioquake Group, 2014].

Kuten koodikatkelmasta 8 nähdään, funktio *MSG_WriteBits()* sisältää nimensäkin mukaisesti runsaasti erilaisia bittitason operaatioita. Koodikatkelman toiminnallisuus voidaan jakaa kolmeen eri osaan: kirjoitettavan kokonaisluvun alustukseen, tavurajan ylimenevien bittien kirjoittamiseen sekä loppujen bittien kirjoittamiseen tavuittain.

Koska *MSG_WriteBits()*-funktio voi saada parametrinaan eripituisia kokonaislukuja, kokonaislukuparametri value on syytä alustaa kuten koodikatkelman rivillä 3 on tehty. Apuna käytetään kolmantena parametrina saatua kirjoitettavan bittijonon pituutta sekä neljätavuista bittijonoa, jossa kaikki bitit ovat päällä. Tätä bittijonoa siirretään oikealle niin, että vain kirjoitettavan bittijonon pituutta vastaava määrä bittejä on päällä. Lopuksi käytetään bittitason JA-operaattoria operandeinaan kirjoitettava bittijono value sekä muodostettu apubittijono. Tuloksena saadaan kokonaisluku, joka vastaa parametrina annettua kirjoitettavaa kokonaislukua muut kuin kirjoitettavat bitit nollattuna.

Seuraavaksi rivillä 4 tarkastetaan, onko kirjoitettava määrä bittejä jaollinen kahdeksalla ja näin ollen kirjoitettavissa tavuittain. Koodikatkelmassa tämä suoritetaan käyttämällä bittitason JA-operaattoria ja vertailulukua seitsemän alakohdassa 2.5.1. havainnollistettuun tapaan. Käytännössä sama tulos saadaan ainakin gcc-kääntäjän versiolla 4.6 [GNU, 2014a]

käyttämällä yksinkertaisesti jakojäännösoperaattoria %. Gcc:n ja as:n [GNU, 2014b] tuottamaa assembly-lähdekoodia ja alkuperäistä c-lähdekoodia vertaileva listaus voidaan tuottaa helposti komennoilla ”gcc -S -fverbose-asm -g tiedosto.c -o tiedosto.s” ja ”as -alhnd tiedosto.s > tiedosto.lst” [Arndt, 2011]. Alun perin näin tuotettu koodikatkelma 9 havainnollistaa operaatioiden vastaavuutta funktiossa *MSG_WriteBits()*, kun jakojäännös tuotetaan eri kääntämiskerroilla operaattorilla & ja operaattorilla %.

```
if (bits % 8) {
1     .loc 1 182 0
2     movl  -48(%rbp), %eax
3     andl  $7, %eax
4     testl %eax, %eax
5     je    .L39
if (bits & 7) {
1     .loc 1 182 0
2     movl  -48(%rbp), %eax
3     andl  $7, %eax
4     testl %eax, %eax
5     je    .L39
```

Koodikatkelma 9. Funktio msg.c : MSG_WriteBits() [The ioquake Group, 2014]. Gcc:n [GNU, 2014a] tuottamat assembly-lähdekoodit eri C-operaattoreita käytettäessä.

Koska käännetyt lähdekoodit vastaavat täysin toisiaan, olisi lähdekoodin luettavuuden nimissä perusteltua suosia jakojäännösoperaattorin käyttöä bittitason operaattorin sijaan. Toisaalta bittitason operaattoria käytettäessä voidaan varmistua ohjelman toimintatavan tehokkuudesta kääntöympäristöstä huolimatta. Asian yksityiskohtaisempi tarkastelu eri kääntäjillä ei kuitenkaan olisi nyt tarkoituksenmukaista, joten siirrytään takaisin funktion *MSG_WriteBits()* tarkasteluun ylemmällä tasolla tiedostaen tämä epäkohta.

Koodikatkelman 8 rivin 4 ehtolausekkeen jälkeisistä riveistä on nähtävissä, että mikäli kirjoitettavan bittijonon pituus ei ole jaollinen kahdeksalla, kirjoitetaan ylimenevät bitit funktiolla *Huff_putBit()*. Tavurajan yli menevät bitit lisätään viestipuskuriin sellaisenaan, sillä Huffmanin koodi muodostetaan tavuittain aakkoston koostuessa yksitavuisista kokonaisluvuista. Funktio *Huff_putBit()* vastaa toiminnallisuudeltaan lähes täysin funktiota *add_bit()*. Ainoana erotuksena on se, että *Huff_putBit()* ottaa yhden lisäparametrin, joka määrittää bittien kirjoituskohdan puskurissa. Bitit kirjoitetaan yksi kerrallaan vähiten merkitsevästä bitistä aloittaen. Tarkastellaan bittien kirjoittamista lähemmin funktion *add_bit()* kuvauksen yhteydessä.

For-silmukasta poistuttaessa tavurajan yli menneet bitit on lisätty kirjoituspuskuriin ja kyseiset bitit poistettu kokonaislukumuuttujasta value. Tämän jälkeen alkaa jäljellä olevien

tavujen kirjoittaminen funktiota *Huff_offsetTransmit()* käyttäen. *Huff_offsetTransmit()* on verrattain kyseenalainen funktio, sillä sen ainoa tarkoitus on asettaa bittien kirjoituskohta puskurissa ja kutsua funktiota *send()* asianmukaisilla parametreilla. Lienee kohtuullista olettaa, että nämä toimenpiteet olisi voitu yhdistää samaan funktioon. *Huff_offsetTransmit()* lähettää funktiolle *send()* parametreina kirjoitettavaa tavua vastaavan solmun ioquake3:n staattisessa Huffmanin puussa sekä osoittimen käytettävään kirjoituspuskuriin.

Funktio *send()* vastaa varsinaisesta Huffmanin koodin muodostamisesta. Funktio käy läpi ioquake3:ssa linkitettynä listana toteutetun Huffmanin puun rekursiivisesti aloittaen parametrina annetusta solmusta. Kirjoituspuskuriin lisätään bitti 0 puuta vasemmalle kuljettaessa ja bitti 1 oikealle kuljettaessa. Kun saavutaan puun juurisolmuun, rekursiosilmukka purkautuu. Toiminta vastaa alakohdassa 3.2.2. esitettyä periaatetta Huffmanin koodia muodostettaessa. Bittien lisäämiseen kirjoituspuskuriin käytetään funktiota *add_bit()*, joka on esitetty alla olevassa koodikatkelmassa 10.

```
1  static void add_bit (char bit, byte *fout) {
2     if ((bloc&7) == 0) {
3        fout[(bloc>>3)] = 0;
5     }
6     fout[(bloc>>3)] |= bit << (bloc&7);
7     bloc++;
8  }
```

Koodikatkelma 10. Funktio huffman.c : add_bit() [The ioquake Group, 2014].

Koodikatkelmasta 10 nähdään, että funktio *add_bit()* tarkistaa aikaisemmin tässä alakohdassa havainnollistettuun tapaan muuttujan bloc jakojäännöksen kahdeksalla jaettaessa. Bloc on funktion ulkopuolinen staattinen muuttuja, joka kertoo Huffmanin koodaukseen liittyville funktioille nykyisen kirjoituskohdan käytetyssä kirjoituspuskurissa. Näin ollen rivillä 2 testataan, onko nykyinen kirjoituskohta tavurajalla. Mikäli näin on, suoritetaan rivillä 3 kuvattu sijoitus.

Kokonaislukumuuttujan bittien siirtäminen kolme kohtaa oikealle on jälleen tapa välttää tavallisen laskutoimitusoperaattorin käyttöä, sillä edellä mainitun operaation tuloksena saadaan tässä tapauksessa vasemman operandin arvo jaettuna kahdeksalla. Saatu arvo kertoo käytännössä seuraavaksi kirjoitettavan tavun järjestysnumeron, jonka avulla saadaan kirjoituspuskurin seuraavan tavun sijainti. Sijoitusoperaation tarkoituksena on yksinkertaisesti nollata kirjoitettavan tavun bitit ennen kirjoittamista.

Rivillä 6 bitti kirjoitetaan osaksi puskurin nykyistä tavua siirtäen ensin bitti haluttuun kohtaan bittijonoa. Bittisiirtojen määrä saadaan jälleen bloc-muuttujan jakojäännöksestä kahdeksalla jaettaessa. Lopuksi mahdollinen päällä oleva bitti lisätään osaksi kirjoituspuskurin nykyistä tavua bittitason TAIlla.

Huomionarvoista on, että tämä periaate toimii vain, kun kirjoitettava tavu on alun perin nollattu ja parametrina saatavan bit-kokonaisluvun arvo on aina joko 0 tai 1. Kuten alakohdassa 2.5.1. havainnollistettiin, bittitason TAI ei koskaan aseta bittejä pois päältä eikä näin ollen aseta bitin arvoksi nollaa, mikäli bitti on ennestään päällä. Koska bittijono on tässä tapauksessa oletuksena nollattu, lisättäviä bittejä ei erikseen tarvitse asettaa pois päältä, kun lisättävän bitin arvo on 0. Periaatteessa rivi 6 voitaisiinkin jättää kokonaan suorittamatta, mikäli lisättävän bitin arvo on 0.

Kun kaikki funktiolle *MSG_WriteBits()* parametrina lähetetyt bitit on kirjoitettu, lähetettävä tieto on koodattu ja valmis eteenpäin lähetettäväksi. Huffmanin koodin purkaminen hoituu täsmälleen päinvastaisella tavalla kuin nyt käsitelty koodin kirjoittaminen, joten siihen käytettäviä funktioita ei liene tarpeen tarkastella lähdekooditasolla.

5.6. Verkkoviiveen kompensointimenetelmät

Ioquake3 on verkkoviiveen kompensointimenetelmiltään melko yllätyksetön, ja se vastaa pitkälti samankaltaisissa peleissä yleisesti käytettävää menetelmää. Kuten kohdassa 3.3. mainittiin, verkkoviiveen vaikutuksia pyritään yleensä minimoimaan erilaisilla yhdistelmillä sijaintien ekstrapolointi- ja interpolointitekniikoita. Ioquake3:n malli perustuu kohdassa 3.3. mainittuun yhdistelmään paikallisen pelaajan sijainnin ekstrapolointia ja muiden pelientiteettien sijaintien interpolointia menneisiin sijainteihin perustuen. Kuten arvata saattaa, sijaintien interpolointi perustuu palvelimelta saatuihin tilannevedoksiin. Mikäli interpolointiin käytettävä tilannevedos jää jostakin syystä vastaanottamatta, ioquake3 käyttää ekstrapolointia interpoloinnissa käytettävän toisen sijainnin määrittämiseksi.

Ioquake3 ei käytä kohdassa 3.3. havainnollistettua menetelmää, jossa palvelin laskee muun muassa ammutut osumat menneisyydessä oleviin sijainteihin perustuen. Sen sijaan ioquake3:ssa täytyy vastustajaa kohti tähdätessä ennakoida vastustajan sijainti tulevaisuudessa, jotta osuma rekisteröidään. Pienenä apuna peli toistaa jokaisen osuman yhteydessä lyhyen äänen, jonka perusteella pelaaja tietää osuvansa [Bernier, 2001]. Tätä

ratkaisua ei voitane pitää korkealatenssisille pelaajille moitteettomana, mutta toisaalta kaikki verkkoviiveen kompensointimenetelmät tuovat mukanaan erilaisia ongelmia.

Sijaintien ekstrapolointiin ja interpolointiin käytettäviä funktioita ei liene mielekästä tarkastella yksityiskohtaisesti, sillä laskelmat ovat pitkälti riippuvaisia pelikohtaisten entiteettien ominaisuuksista. Erilaiset verkkoviiveen kompensointimenetelmät ovat olleet historiallisesti suuri osa internet-pelien kehitystä, ja näin ollen niitä kuvaavaa dokumentaatiota on olemassa myös melko kattavasti. Esimerkiksi Bernier [2001] sekä Armitage ja muut [2006] esittelevät vieläkin yleisesti käytössä olevia verkkoviiveen kompensointimenetelmiä ja niiden toteutusta varsin yksityiskohtaisesti. Myös Sanglard [2009] on käsitellyt erityisesti sijaintien ekstrapoloinnin toteutusta Quakessa.

6. Kehitysehdotuksia

Ioquake3:n verkkotekniikan yksityiskohtainen tarkastelu voidaan saattaa päätökseen pohtimalla mahdollisia kehityssuuntauksia ja niiden toteutusta käsitellyn verkkoprotokollan pohjalle. Koska ioquake3:n ja sen edeltäjien verkkotekniikkaa on hiottu jo vuosia, ei protokollan jatkokehittämistä tässä vaiheessa voida pitää yksinkertaisena tehtävänä. Luvussa 5 tehty yksityiskohtainen tarkastelu paljasti kuitenkin muutamia mahdollisia kehityskohteita, joita voidaan havainnollistaa lähdekoodia muokkaamalla (ks. liite 2) tai yleisemmän tason huomioilla. Tässä kohdassa mainittavia kehitysehdotuksia ei voida pitää verkkotekniikan kokonaisarkkitehtuurin kannalta kovinkaan merkityksellisinä, mutta tästä huolimatta tehtävillä muutoksilla voidaan saavuttaa joitakin positiivisia tuloksia.

6.1. Merkkijonomuotoiset viestit

Kuten muun muassa alakohdissa 5.4.1. ja 5.4.3. havaittiin, ioquake3:n lähettämät komennot ovat kaistan ulkopuolisten viestien ja luotettavien viestien tapauksessa yleensä pitkiä merkkijonoja. Tätä ei voida pitää verkkoliikenteen minimoinnin kannalta tehokkaimpana mahdollisena ratkaisuna, sillä periaatteessa merkkijonot voisi korvata esimerkiksi yksi- tai muutamatavuisilla tunnisteilla. Pelin aikana lähetettävät epäluotettavat viestit toimivatkin yleensä tällä tekniikalla.

Luotettavien ja kaistan ulkopuolisten viestien lähettämiselle merkkijonoina ei liene olemassa erityisen pätevää syytä, sillä edes viestien selkokielisyys verkkoliikennettä tutkiessa ei Huffmanin koodauksen ansiosta merkitse mitään. Myöskään lähdekoodin luettavuuden kannalta asialla ei ole merkitystä, sillä jokaista komentoa vastaavalle bittijonolle voidaan määritellä selkokielinen nimi esimerkiksi enum-tyypin avulla.

Merkkijonokomentojen muuttaminen yksitavuisiksi komennoiksi vaatii joukon muutoksia ioquake3:n lähdekoodiin, mutta periaatteessa tätä toimenpidettä voitaneen pitää melko yksinkertaisena. Liitteessä 2 listatut muutokset lähdekooditiedostoon q_shared.h havainnollistavat erästä tapaa määrittää yksitavuiset komennot. Kun komennot määritetään enum-tyyppinä, niiden merkitys on tulkittavissa samaan tapaan kuin merkkijonoihin pohjautuvienkin komentojen. Liitteessä 2 lähdekooditiedoston sv_client.c funktiossa *SV_GetChallenge()* lähetettävät kaistan ulkopuoliset viestit havainnollistavat yksitavuisten komentojen käyttöä vanhojen merkkijonojen sijaan. Kuten kyseisestäkin funktiosta voi päätellä, komentojen muuttaminen yksitavuisiksi vaatii pieniä mutta kriittisiä muutoksia hyvin laajalla alueella lähdekoodia.

Säästeliäämmän tilankäytön lisäksi kokonaislukumuotoisten komentojen käyttäminen mahdollistaa komentojen tunnistamisen esimerkiksi switch-lausetta käyttäen. Kuten liitteen 2 muutoksista nähdään, switch-lauseen käyttöä on nopeuden lisäksi pidettävä myös lähdekoodin luettavuuden kannalta parempana ratkaisuna kuin vanhaa merkkijonojen vertailuun perustuvaa ratkaisua.

6.2. Mukautuva Huffmanin koodaus

Kuten alakohdassa 5.5.3. mainittiin, ioquake3 tukee myös mukautuvaa Huffmanin koodausta, mutta tästä huolimatta sitä käytetään koko verkkokoodissa ainoastaan yhden verkkoyhteyden muodostamiseen liittyvän viestin (”connect ...”) koodaukseen. Tätä on pidettävä merkillisenä ratkaisuna, sillä mukautuvaa Huffmanin koodausta voitaisiin sujuvasti käyttää muidenkin tietyt kriteerit täyttävien viestien koodaukseen.

Kuten liitteessä 1 esitetystä ioquake3:n staattisesta Huffmanin puusta nähdään, muun muassa aakkosia vastaavat koodisanat ovat pitkiä – paikoin jopa pitempiä kuin merkin koodaamaton pituus eli yksi tavu. Tämä lienee myös syy mukautuvan Huffmanin koodauksen käyttöön aiemmin mainitussa yhteydenmuodostusviestissä, sillä staattiseen puuhun perustuva koodaus tuottanee kyseisen viestin tapauksessa alkuperäistä pitemmän bittijonon. Mukautuvan Huffmanin koodauksen voitaneen katsoa olevan yleisestikin käyttökelpoinen ratkaisu pitkien selkokielisten merkkijonojen koodaukseen ioquake3:n staattiseen Huffmanin puuhun perustuvan koodauksen sijaan.

Muun muassa pelaajien pelin aikana lähetettämät keskusteluviestit voisivat olla mielekäs kohde mukautuvalle Huffmanin koodaukselle. Esimerkiksi tavallisessa englanninkielisessä lauseessa esiintyvä merkkihajonta näyttänee varsin erilaiselta ioquake3:n lähettämään keskimääräiseen tietoon verrattuna. Myös kaikki ”connect”-komennon kaltaiset pitkiä merkkijonoja parametrinaan saavat komennot voisi koodata mukautuvalla Huffmanin koodauksella.

Koska mukautuvaa Huffmanin koodausta käytetään ioquake3:ssa vain yhdessä erikoistapauksessa, vaatii sen käyttöönotto laajemmin jonkin verran muutoksia lähdekoodiin. Toimintaperiaate voidaan yhdistää edellisessä kohdassa lisättyihin yksitavuisiin komentoihin määrittämällä jokin komentoa vastaavan bittijonon bitti niin sanotuksi ”Huffmanin bitiksi”. Mikäli määritetty Huffmanin bitti on päällä, suoritetaan viestin koodaus tai purkaminen mukautuvalla Huffmanin koodauksella. Tämä

mahdollistaa koodauksen määrittämisen jopa viestikohtaisesti, sillä sama komento voidaan lähettää joko Huffmanin bitti päällä tai pois päältä.

Koska ioquake3:ssa käytettävien komentojen kirjo on varsin suppea, voidaan Huffmanin bitin arvoksi määrittää esimerkiksi yksitavuisen bittijonon merkitsevin bitti. Tämä bitti on päällä ainoastaan, kun bittijonoa vastaava kokonaisluku on 128 tai suurempi. Näin ollen Huffmanin bitti voidaan määrittää liitteen 2 kohdassa q_shared.h kuvatulla tavalla. ADAPTIVE_HUFFMAN-bitin käyttöä havainnollistetaan liitteessä 2 muun muassa funktiossa *SV_ConnectionlessPacket()*. Bitin tila voidaan tarkistaa bittitason JAlla ja bitin pois päältä kytkeminen bittitason JAn ja 1:n komplementin yhdistelmällä. Kuten tavallisesti, bitin päälle asettaminen onnistuu bittitason TAIlla.

Jotta mukautuvalla Huffmanin koodauksella muodostettu viesti osataan purkaa oikein, täytyy muun muassa funktioihin *CL_ConnectionlessPacket()* ja *SV_ConnectionlessPacket()* lisätä tarkistukset Huffmanin bitin varalta. Niin ikään tavallista Huffmanin koodausta käyttävien luku- ja kirjoitusfunktioiden toimintaa on syytä ohjata esimerkiksi viestitietueeseen lisättävän muuttujan avulla, jotta viestiä ei vahingossa koodata tai pureta kahteen kertaan eri koodaustekniikoilla.

Mukautuvan Huffmanin koodauksen käytön mahdollistavien muutoksien ei voida katsoa monimutkaistavan lähdekoodia merkittävästi, sillä suurin osa vaadittavasta toiminnallisuudesta on ioquake3:ssa jo valmiina. Mahdollisuus käyttää joko tavallista tai mukautuvaa Huffmanin koodausta voidaan toteuttaa melko saumattomasti, eikä tämän toiminnallisuuden lisäämättä jättämiseen liene olemassa mitään painavaa syytä. Kaikki tarvittavat toimenpiteet on toteutettu liitteessä 2 kuvatuissa muutoksissa.

6.3. Lähdekoodin luettavuus

Alakohdassa 5.5.3. havainnollistetun bittitason operaatioiden melko kryptisen käytön voidaan katsoa heikentävän lähdekoodin luettavuutta merkittävissä määrin. Tämä olisi puolusteltavissa, mikäli nykyaikaiset kääntäjät eivät testattuun tapaan muokkaisi lähdekoodia tehokkaampaan muotoon automaattisesti. Koska operaatioiden kääntämistä ei testattu muuta kuin gcc-kääntäjällä [GNU, 2014a], ei bittitason operaatioiden muuttamista vastaaviksi laskuoperaatioiksi voi varauksetta osoittaa paremmaksi ratkaisuksi. On kuitenkin selvää, että varsinkin avoimen lähdekoodin projekteissa lähdekoodin luettavuus on merkityksellinen asia. Mikäli vastaava tehokkuus ja parempi luettavuus voidaan yhdistää, ei muutosten tekemiselle pitäisi olla estettä.

7. Loppusanat

Kuten kohdassa 3.4. tiivistettiin, reaaliaikaisten internet-pelien suurimmat haasteet liittyvät verkkoviiveen vaikutusten ja verkkoliikenteen minimoimiseen. Tämän lisäksi internet-pelien verkkoprotokollien on toteutettava joukko tavanomaisia sovelluskerroksen protokollan ominaisuuksia kuten luotettavuus ja yhteydellisyys. Muun muassa luvussa 5 tehdyn yksityiskohtaisen tarkastelun perusteella voitaneen kuitenkin sanoa, että tavanomaisempienkin ominaisuuksien toteuttamisessa on otettava huomioon reaaliaikaisten internet-pelien erityistarpeet. Esimerkiksi valikoivan luotettavuuden lisääminen protokollaan saattaa helposti aiheuttaa virheenhallinnan kannalta ongelmallisia tilanteita. Tällaisten ongelmien välttämiseksi toimivaksi todettujen verkkotekniikkaratkaisujen tutkiminen on oiva apukeino.

Ioquake3:n voidaan katsoa täyttävän tavalla tai toisella kaikki reaaliaikaisen internet-pelin verkkoprotokollan keskeiset vaatimukset: verkkoviiveen vaikutusten ja verkkoliikenteen minimoinnin sekä protokollan yhteydellisyyden ja valikoivan luotettavuuden. Ioquake3 lienee vahvimmillaan erityisesti verkkoliikenteen minimoinnin saralla, sillä tiedon pakkaukseen käytetyt tekniikat ovat tarkoituksenmukaisia ja tehokkaita. Verkkoprotokollan vahvuuksiin kuuluu myös tehokas matalan tason toteutus, joka on hyvin pitkälle optimoitu. Ioquake3:n verkkoprotokollan lähdekoodia voidaankin pitää kattavana esimerkkikokoelmana erilaisia bittitason ohjelmointitekniikoita C-kielellä. Kuten luvun 5 tarjoamista esimerkeistä nähdään, bittitason ohjelmointitekniikat ovat välttämätön apu sovelluskerroksen verkkoprotokollan ohjelmoinnissa.

Ioquake3:n toteuttamaa pelientiteettien sijaintien arviointia ekstrapolointia ja interpolointia käyttäen voitaneen pitää nykyään tavanomaisena verkkoviiveen torjuntamekanismina. Koska kaikilla torjuntamekanismeilla on omat haittapuolensa, paras ratkaisu on aina riippuvainen pelin luonteesta. Toisin kuin esimerkiksi tehokkaat verkkopaketin pakkaustekniikat, verkkoviiveen torjuntatekniikat eivät ole sovellettavissa sellaisenaan kaikille verkkopeleille. Uusimpien kaupallisten internet-pelien lähdekoodin ollessa käytännössä poikkeuksetta suljettua aihealueella tapahtunut mahdollinen kehitys on sekin toistaiseksi pimennon peitossa. Aihealueen kirjallisuudessakaan ei juuri esitetä ioquake3:n tekniikoista poikkeavia ratkaisuja, joten keinotekoisen viiveen sekä sijaintien interpoloinnin ja ekstrapoloinnin yhdistelmää voitaneen yhä pitää nykyaikaisena verkkoteknisenä ratkaisuna.

Koska ioquake3:a kehitetään edelleen, sen voidaan katsoa olevan oiva alusta reaaliaikaisiin internet-peleihin liittyviin jatkotutkimusprojekteihin. Aikaisempiin tutkimuksiin [Sanglard, 2012; Stefyn *et al.*, 2011; Ploss *et al.*, 2008; Armitage, 2003] ja nyt tehtyyn analyysiin perustuva tietopohja tarjoaa helposti lähestyttävän rajapinnan pelilajin verkkotekniikkaan. Tässä tutkimuksessa käsitellyt ioquake3:n ominaisuudet ovat lisäksi sovellettavissa laajaan kirjoon erilaisia internet-pelejä, sillä erityisesti tehokkaasti toteutetut pakkausmenetelmät ovat tuskin koskaan epätoivottuja.

Viiteluettelo

[Abrash, 1997a] Michael Abrash, *Graphics Programming Black Book.* Coriolis Group Books, 1997.

[Abrash, 1997b] Michael Abrash, Quake's 3-D engine – the big picture. *Dr. Dobb's Journal*, **22,** 19 (Spring 1997).

[Accardo, 2004] Sal Accardo, *DOOM 3*. GameSpy, http://pc.gamespy.com/pc/doom-3/536705p1.html (retrieved 13.2.2014).

[Adams, 2004] Dan Adams, *DOOM 3 review*. IGN, http://uk.ign.com/articles/2004/08/06/doom-3-review (retrieved 13.2.2014).

[Aggarwal *et al.*, 2004] Suhid Aggarwal, Hemant Banavar, Amit Khandelwal, Sarit Mukherjee and Sampath Rangarajan. Accuracy in dead-reckoning based distributed multi-player games. In: *Proc. of the 3rd SIGCOMM Workshop on Network and System Support for Games (NetGames '04)*, 161-165.

[Airey *et al.*, 1990] John M. Airey, John H. Rohlf and Frederick P. Brooks, Jr., Towards image realism with interactive update rates in complex virtual building environments. In: *Proc. of the 1990 Symposium on Interactive 3D Graphics*, 41-50.

[AMD, 2013] Advanced Micro Devices, *AMD64 Architecture Programmer's Manual Volume 1: Application Programming.* Available as: http://developer.amd.com/wordpress/media/2012/10/24592_APM_v11.pdf (retrieved 13.2.2014).

[Armitage, 2003] Grenville Armitage, An experimental estimation of latency sensitivity in multiplayer Quake 3, Swinburne University of Technology, Centre for Advanced Internet Architectures, Tech. Rep. 030405A, Apr. 2003.

[Armitage *et al.*, 2006] Grenville Armitage, Mark Claypool and Philip Branch, *Networking and Multiplayer Games: Understanding and Engineering Multiplayer Internet Games*. Wiley, 2006.

[Armitage, 2007] Grenville Armitage, Server discovery for Quake III Arena, Swinburne University of Technology, Centre for Advanced Internet Architectures, Tech. Rep. 110209A, Jul. 2007.

[Arndt, 2011] Jörn Arndt, *Matters Computational: Ideas, Algorithms, Source Code*. Springer, 2011.

[Beigbeder *et al.*, 2004] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu and Mark Claypool, The effects of loss and latency on user performance in Unreal Tournament 2003®. In: *Proc. of the 3rd ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '03),* 144-151.

[Bernier, 2001] Yahn W. Bernier, Latency compensating methods in client/server in-game protocol design and optimization. In: *Proc. of the 15th Game Developers Conference*, 73-85.

[Bharambe *et al.*, 2006] Ashwin Bharambe, Jeffrey Pang and Srinivasan Seshan, Colyseus: a distributed architecture for online multiplayer games. In: *Proc. of the International Conference on Networked Systems Design and Implementation (NSDI '06)*, 155-168.

[Bharambe *et al.*, 2008] Ashwin Bharambe, John R. Douceur, Jacob R. Lorch, Thomas Moscibroda, Jeffrey Pang, Srinivasan Seshan and Xinyu Zhuang, Donnybrook: enabling large-scale, high-speed, peer-to-peer games. In: *Proc. of the ACM SIGCOMM 2008 Conference on Data communication*, 389-400.

[Braden, 1989] Robert Braden (ed.), Requirements for internet hosts – communication layers. STD 3, RFC 1122, USC/Information Sciences Institute, Oct 1989. Available as: http://tools.ietf.org/html/rfc1122 (retrieved 13.2.2014).

[Bonham *et al.*, 2000] Shawn Bonham, Daniel Grossman, William Portnoy and Kenneth Tam, Quake: An Example Multi-User Network Application – Problems and Solutions in Distributed Interactive Simulations. University of Washington, CSE 561 Term Project Report, May 2000.

[Boulanger *et al.*, 2006] Jean-Sébastien Boulanger, Jörg Kienzle and Clark Verbrugge, Comparing interest management algorithms for massively multiplayer games. In: *Proc. of the 5th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '06)*, article no. 6.

[Carmack, 1996] John Carmack, *Here is the new plan.* http://fabiensanglard.net/quakeSource/johnc-log.aug.htm (retrieved 13.2.2014).

[Claypool, 2008] Mark Claypool, Network characteristics for server selection in online games. In: *Proc. of the 15th Annual Multimedia Computing and Networking Conference (MMCN '08*).

[Cohen, 1981] Danny Cohen, On holy wars and a plea for peace. *IEEE Computer Magazine*, **14**, 10 (Oct. 1981), 48-54.

[Comer, 2000] Douglas E. Comer, *Internetworking with TCP/IP: Principles, Protocols and Architectures.* Prentice Hall, 2000.

[Day, 1995] John Day, The (un)revised OSI reference model. *ACM SIGCOMM Computer Communication Review*, **25**, 5 (Oct. 1995), 39-55.

[Debroy *et al.*, 2013] Saptarshi Debroy, Mohammad Zubair Ahmad, Mukundan Iyengar and Mainak Chatterjee, Critical sections in networked games. In: *Proc. of IEEE International Conference on Communications (ICC 2013)*, 2365-2369.

[Diot & Gautier, 1999] Christophe Diot and Laurent Gautier, A distributed architecture for multiplayer interactive applications on the internet. *IEEE Network Magazine*, **13**, 4 (Jul./Aug. 1999), 6-15.

[GameTracker, 2014] GameTracker, http://www.gametracker.com/ (retrieved 13.2.2014).

[Garfinkel *et al.*, 2003] Simson Garfinkel, Gene Spafford and Alan Schwartz, *Practical Unix & Internet Security, 3rd Edition*. O'Reilly Media, 2003.

[GNU, 2014a] GNU, *The GNU Compiler Collection*, http://gcc.gnu.org (retrieved 13.2.2014).

[GNU, 2014b] GNU, *The GNU Binutils*, http://www.gnu.org/software/binutils/ (retrieved 13.2.2014).

[Guo *et al.*, 2003] Katherine Guo, Sarit Mukherjee, Sampath Rangarajan and Sanjoy Paul, A fair message exchange framework for distributed multi-player games. In: *Proc. of the 2nd Workshop on Network and System Support for Games (NetGames '03)*, 29-41.

[Harcsik *et al.*, 2007] Szabolcs Harcsik, Andreas Petlund, Carsten Griwodz and Pål Halvorsen, Latency evaluation of networking mechanisms for game traffic. In: *Proc. of the 6th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '07)*, 129-134.

[Harris & Harris, 2012] David Harris and Sarah Harris, *Digital Design and Computer Architecture, 2nd Edition*. Morgan Kauffmann, 2012.

[Hook, 2004] Brian Hook, *The Quake3 networking model*. http://trac.bookofhook.com/bookofhook/trac.cgi/wiki/Quake3Networking (retrieved 13.2.2014).

[Huang & Griffioen, 2013] Shufeng Huang and James Griffioen, HyperNet games: leveraging SDN networks to improve multiplayer online games. In: *Proc. of the 18th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games (CGAMES 2013)*, 74-78.

[Id Software, 1997] Id Software, *Doom source code*. https://github.com/id-Software/DOOM (retrieved 13.2.2014).

[Id Software, 1999] Id Software, *Quake source code*. https://github.com/id-Software/Quake (retrieved 13.2.2014).

[Id Software, 2001] Id Software, *Quake 2 source code*. https://github.com/id-Software/Quake-2 (retrieved 31.7.2013).

[Id Software, 2005] Id Software, *Quake III Arena source code*. https://github.com/id-Software/Quake-III-Arena (retrieved 13.2.2014).

[Id Software, 2011] Id Software, *Doom 3 source code*. https://github.com/id-Software/DOOM-3 (retrieved 13.2.2014).

[Id Software, 2014] Id Software, *Quake Live.* http://www.quakelive.com/ (retrieved 13.2.2014)

[Intel, 2013] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture*. Available as: http://download.intel.com/design/processor/manuals/253665.pdf (retrieved 13.2.2014).

[Isensee, 2004] Pete Isensee, Bit packing: a network compression technique. In: Andrew Kirmse (ed.), *Game Programming Gems 4*, Charles River Media, 2004, 571-578.

[ISO, 1994] International Organization for Standardization, *Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*, ISO/IEC 7498-1:1994, 1994. Available as: http://www.ecma-international.org/activities/Communications/TG11/s020269e.pdf (retrieved 13.2.2014).

[ISO, 2011] International Organization for Standardization, *Information Technology – Programming Languages – C,* ISO/IEC 9899:2011, 2011.

[Kernighan & Ritchie, 1988] Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language, 2nd Edition.* Prentice Hall, 1988.

[Kushner, 2004] David Kushner, *Masters of Doom.* Random House, 2004.

[Li *et al.*, 2011] Yadong Li, Wenqiang Cui, Danlan Li and Rui Zhang, Research based on OSI model. In: *Proc. of the 3rd IEEE Conference on Communication Software and Networks (ICCSN 2011*), 554-557.

[Maathuis & Smit, 2003] Ivo Maathuis and Wim A. Smit, The battle between standards: TCP/IP vs OSI victory through path dependency or by quality? In: *Proc. of the 3rd IEEE Conference on Standardization and Innovation in Information Technology (SIIT 2003)*, 161-176.

[Maiberg, 2014] Emanuel Maiberg, *Quake Live gets standalone client*. GameSpot, http://www.gamespot.com/articles/quake-live-gets-standalone-client/1100-6417080/ (retrieved 13.2.2014).

[Mäki-Panula, 2001] Jouni Mäki-Panula, *Qtest 5 vuotta.* http://dome.fi/pelit/artikkelit/pelit/qtest-5-vuotta (tarkistettu 13.2.2014).

[Pantel & Wolf, 2002] Lothar Pantel and Lars C. Wolf, On the impact of delay on real-time multiplayer games. In: *Proc. of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '02),* 23-29.

[Pavlicic & Armitage, 2003] Ana M. Pavlicic, Grenville Armitage, Quake 3 packet inter-arrival and length over the internet, Swinburne University of Technology, Centre for Advanced Internet Architectures, Tech. Rep. 030919B, Sept. 2003.

[Ploss *et al.*, 2008] Alexander Ploss, Stefan Wichmann, Frank Glinka and Sergei Gorlatch, From a single- to multi-server online game: a Quake 3 case study using RTF. In:

Proc. of the 2008 International Conference on Advances in Computer Entertainment Technology (ACE '08), 83-90.

[Pozzobon, 2002] Mark Pozzobon, Quake 3 packet and traffic characteristics, Swinburne University of Technology, Centre for Advanced Internet Architectures, Tech. Rep. 021220A, Dec. 2002.

[Quax *et al.*, 2004] Peter Quax, Patrick Monsieurs, Wim Lamotte, Danny De Vieeschauwer and Natalie Degrande. Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game. In: *Proc. of 3rd ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '04)*, 152-156.

[Reynolds & Postel, 1994] Joyce K. Reynolds and Jon Postel, *Assigned numbers*. IETF STD 3, RFC 1700, USC/Information Sciences Institute, Oct. 1994. Available as: http://www.ietf.org/rfc/rfc1700.txt (retrieved13.2.2014).

[Romero, 1996] John Romero, *The official Quake deathmatch test, operating instructions.* http://www.quaketerminus.com/tools/qtest1.zip (retrieved 13.2.2014).

[Sanglard, 2009] Fabien Sanglard, *Quake engine code review,* http://fabiensanglard.net/quakeSource/index.php (retrieved 13.2.2014).

[Sanglard, 2012] Fabien Sanglard, *Quake 3 source code review*, http://fabiensanglard.net/quake3/index.php (retrieved 13.2.2014).

[Sayood, 2012] Khalid Sayood, *Introduction to Data Compression, 4th Edition*, Morgan Kauffmann, 2012.

[Smed *et al.*, 2002a] Jouni Smed, Timo Kaukoranta and Harri Hakonen, Aspects of networking in multiplayer computer games. *The Electronic Library*, **20**, 2, 87-97.

[Smed *et al.*, 2002b] Jouni Smed, Timo Kaukoranta and Harri Hakonen, A review on networking and multiplayer games. University of Turku, Turku Centre for Computer Science, Tech. Rep. 454, Apr. 2002.

[Stefyn *et al.*, 2011] D. Stefyn, A.L. Cricenti and P.A. Branch, Quake III Arena game structures, Swinburne University of Technology, Centre for Advanced Internet Architectures, Tech. Rep. 110209A, Feb. 2011.

[Stevens, 1993] W. Richard Stevens, *TCP/IP Illustrated, Vol. 1: The Protocols*. Addison-Wesley, 1993.

[Stevens *et al.*, 2003] W. Richard Stevens, Bill Fenner and Andrew M. Rudoff, *Unix Network Programming, Volume 1: The Sockets Networking API (3rd Edition).* Addison-Wesley Professional, 2003.

[Stowell *et al.*, 2011] Tim Stowell, Jon Scoresby, K. Chad Coats, Michael R. Capell and Brett E. Shelton, Leveraging open source technology in 3D game engine

development. In: Richard Ferdig (ed.), *Discoveries in Gaming and Computer-Mediated Simulations: New Interdisciplinary Applications*. IGI Global, 2011.

[The ioquake Group, 2014] The ioquake Group, *ioquake3 source code.* https://github.com/ioquake/ioq3 (retrieved 13.2.2014).

[van Waveren, 2007] Jean-Paul van Waveren, Run-Length Compression of Large Sparse Potential Visible Sets, Id Software, 2007. Available as: http://mrelusive.com/publications/papers/Run-Length-Compression-of-Large-Sparse-Potential-Visible-Sets.pdf (retrieved 13.2.2014).

[van Waveren, 2006] Jean-Paul van Waveren, *The Doom III Network Architecture.* Id Software, 2006. Available as: http://mrelusive.com/publications/papers/The-III-Network-Architecture.pdf (retrieved 13.2.2014).

[Yahyavi & Kemme, 2013] Amir Yahyavi and Bettina Kemme, Peer-to-peer architectures for massively multiplayer online games: a survey. *J. ACM Comput. Surv. (CSUR),* **46**, 1 (Oct. 2013), article no. 9.

[Yau & Lam, 1998] David K. Yau and Simon S. Lam, Migrating sockets – end system support for networking with quality of services guarantees. *J. IEEE/ACM Transactions on Networking (TON),* **6**, 6 (Dec. 1998), 700-716.

[Zander & Armitage, 2004] Sebastian Zander and Grenville Armitage, Empirically measuring the QoS sensitivity of interactive online game players. In: *Proc. of the Australian Telecommunications Networks and Applications Conference (ATNAC 2004)*, 511-518.

Liite 1: Ioquake3:n staattinen Huffmanin puu

Kokonaisluku	Paino	Suhdeluku	Koodisana	Merkki
0	250315	23,76%	01	
1	41193	3,91%	11011	
2	6292	0,60%	0001001	
3	7106	0,67%	0011011	
4	3730	0,35%	10000101	
5	3750	0,36%	10001000	
6	6110	0,58%	0000100	
7	23283	2,21%	111111	
8	33317	3,16%	10101	
9	6950	0,66%	0010110	
10	7838	0,74%	1001011	
11	9714	0,92%	1101000	
12	9257	0,88%	1100100	
13	17259	1,64%	101101	
14	3949	0,37%	10011100	
15	1778	0,17%	001101010	
16	8288	0,79%	1010010	
17	1604	0,15%	000110100	
18	1590	0,15%	000011111	
19	1663	0,16%	000111111	
20	1100	0,10%	1011101110	
21	1213	0,12%	1100111111	
22	1238	0,12%	1101010001	
23	1134	0,11%	1100010011	
24	1749	0,17%	001011110	
25	1059	0,10%	1011000110	
26	1246	0,12%	1101010100	
27	1149	0,11%	1100011011	
28	1273	0,12%	1101011110	

29	4486	0,43%	11000010	
30	2805	0,27%	111100100	
31	3472	0,33%	00101011	
32	21819	2,07%	111011	Välilyönti
33	1159	0,11%	1100101100	!
34	1670	0,16%	001000100	”
35	1066	0,10%	1011001101	#
36	1043	0,10%	1010001111	$
37	1012	0,10%	1001111111	%
38	1053	0,10%	1011000011	&
39	1070	0,10%	1011001111	'
40	1726	0,16%	001010010	(
41	888	0,08%	0011010110	)
42	1180	0,11%	1100110000	*
43	850	0,08%	0010100000	+
44	960	0,09%	1000111001	,
45	780	0,07%	0000111101	-
46	1752	0,17%	001011111	.
47	3296	0,31%	00011110	/
48	10630	1,01%	1110010	0
49	4514	0,43%	11000011	1
50	5881	0,56%	11110111	2
51	2685	0,25%	111010011	3
52	4650	0,44%	11001010	4
53	3837	0,36%	10001101	5
54	2093	0,20%	101100000	6
55	1867	0,18%	100001101	7
56	2584	0,25%	111000000	8
57	1949	0,19%	100011110	9
58	1972	0,19%	100110011	:
59	940	0,09%	1000100110	;

60	1134	0,11%	1100010010	<
61	1788	0,17%	001111010	=
62	1670	0,16%	001000101	>
63	1206	0,11%	1100111110	?
64	5719	0,54%	11110011	@
65	6128	0,58%	0000101	A
66	7222	0,69%	0011111	B
67	6654	0,63%	0010000	C
68	3710	0,35%	10000100	D
69	3795	0,36%	10001010	E
70	1492	0,14%	000000010	F
71	1524	0,14%	000011100	G
72	2215	0,21%	101111001	H
73	1140	0,11%	000000010	I
74	1355	0,13%	1110101101	J
75	971	0,09%	1000111011	K
76	2180	0,21%	101110011	L
77	1248	0,12%	1101010101	M
78	1328	0,13%	1110001110	N
79	1195	0,11%	1100110011	O
80	1770	0,17%	001101001	P
81	1078	0,10%	1011100100	Q
82	1264	0,12%	1101011010	R
83	1266	0,12%	1101011011	S
84	1168	0,11%	1100101110	T
85	965	0,09%	1000111010	U
86	1155	0,11%	1100011110	V
87	1186	0,11%	1100110001	W
88	1347	0,13%	1110101100	X
89	1228	0,12%	1101001101	Y
90	1529	0,15%	000011101	Z

91	1600	0,15%	000101010	[
92	2617	0,25%	111000110	\
93	2048	0,19%	101000001	]
94	2546	0,24%	110101110	^
95	3275	0,31%	00011011	_
96	2410	0,23%	110011110	`
97	3585	0,34%	00111100	a
98	2504	0,24%	110101011	b
99	2800	0,27%	111100011	c
100	2675	0,25%	111010010	d
101	6146	0,58%	0000110	e
102	3663	0,35%	10000010	f
103	2840	0,27%	111100101	g
104	14253	1,35%	001110	h
105	3164	0,30%	00010100	i
106	2221	0,21%	101111010	j
107	1687	0,16%	001000110	k
108	3208	0,30%	00011001	l
109	2739	0,26%	111100010	m
110	3512	0,33%	00110000	n
111	4796	0,46%	11001101	o
112	4091	0,39%	10100001	p
113	3515	0,33%	00110001	q
114	5288	0,50%	11100010	r
115	4016	0,38%	10011110	s
116	7937	0,75%	1001101	t
117	6031	0,57%	0000001	u
118	5360	0,51%	11101000	v
119	3924	0,37%	10011000	w
120	4892	0,46%	11010010	x
121	3743	0,36%	10000111	y

122	4566	0,43%	11000101	z
123	4807	0,46%	11001110	{
124	5852	0,56%	11110110	|
125	6400	0,61%	0001011	}
126	6225	0,59%	0001000	~
127	8291	0,79%	1010011	
128	23243	2,21%	111110	
129	7838	0,74%	1001010	
130	7073	0,67%	0011001	
131	8935	0,85%	1011111	
132	5437	0,52%	11110000	
133	4483	0,43%	11000001	
134	3641	0,35%	10000000	
135	5256	0,50%	11100001	
136	5312	0,50%	11100110	
137	5328	0,51%	11100111	
138	5370	0,51%	11101010	
139	3492	0,33%	00101110	
140	2458	0,23%	110100111	
141	1694	0,16%	001000111	
142	1821	0,17%	100000010	
143	2121	0,20%	101100100	
144	1916	0,18%	100011001	
145	1149	0,11%	1100011010	
146	1516	0,14%	000000011	
147	1367	0,13%	1110101111	
148	1236	0,12%	1101010000	
149	1029	0,10%	1010001010	
150	1258	0,12%	1101011000	
151	1104	0,10%	1011110000	
152	1245	0,12%	1101010011	

153	1006	0,10%	1001111100	
154	1149	0,11%	1100011100	
155	1025	0,10%	1010001001	
156	1241	0,12%	1101010010	
157	952	0,09%	1000110000	
158	1287	0,12%	1101011111	
159	997	0,09%	1001110110	
160	1713	0,16%	001010001	
161	1009	0,10%	1001111110	
162	1187	0,11%	1100110010	
163	879	0,08%	0011010000	
164	1099	0,10%	1011101101	
165	929	0,09%	1000001110	
166	1078	0,10%	1011100011	
167	951	0,09%	1000101110	
168	1656	0,16%	000111001	
169	930	0,09%	1000011000	
170	1153	0,11%	1100011101	
171	1030	0,10%	1010001011	
172	1262	0,12%	1101011001	
173	1062	0,10%	1011001100	
174	1214	0,12%	1101001100	
175	1060	0,10%	1011001010	
176	1621	0,15%	000111000	
177	930	0,09%	1000001111	
178	1106	0,10%	1011110001	
179	912	0,09%	0011110111	
180	1034	0,10%	1010001100	
181	892	0,08%	0011010111	
182	1158	0,11%	1100011111	
183	990	0,09%	1001100101	

184	1175	0,11%	1100101111	
185	850	0,08%	0010100001	
186	1121	0,11%	1100000001	
187	903	0,09%	0011110110	
188	1087	0,10%	1011100101	
189	920	0,09%	1000000111	
190	1144	0,11%	1100011001	
191	1056	0,10%	1011000100	
192	3462	0,33%	00101010	
193	2240	0,21%	110000001	
194	4397	0,42%	10111010	
195	12136	1,15%	000001	
196	7758	0,74%	1001000	
197	1345	0,13%	1110001111	
198	1307	0,12%	1110000011	
199	3278	0,31%	00011101	
200	1950	0,19%	100011111	
201	886	0,08%	0011010001	
202	1023	0,10%	1010001000	
203	1112	0,11%	1100000000	
204	1077	0,10%	1011100010	
205	1042	0,10%	1010001110	
206	1061	0,10%	1011001011	
207	1071	0,10%	1011100000	
208	1484	0,14%	000000000	
209	1001	0,10%	1001110111	
210	1096	0,10%	1011101100	
211	915	0,09%	1000000110	
212	1052	0,10%	1011000010	
213	995	0,09%	1001110100	
214	1070	0,10%	1011001110	

215	876	0,08%	0010100111	
216	1111	0,11%	1011110110	
217	851	0,08%	0010100110	
218	1059	0,10%	1011000111	
219	805	0,08%	0001101010	
220	1112	0,11%	1011110111	
221	923	0,09%	1000001100	
222	1103	0,10%	1011101111	
223	817	0,08%	0001111100	
224	1899	0,18%	100010110	
225	1872	0,18%	100010010	
226	976	0,09%	1001100100	
227	841	0,08%	0001111101	
228	1127	0,11%	1100010000	
229	956	0,09%	1000111000	
230	1159	0,11%	1100101101	
231	950	0,09%	1000100111	
232	7791	0,74%	1001001	
233	954	0,09%	1000110001	
234	1289	0,12%	1110000010	
235	933	0,09%	1000011001	
236	1127	0,11%	1100010001	
237	3207	0,30%	00011000	
238	1020	0,10%	1010000001	
239	927	0,09%	1000001101	
240	1355	0,13%	1110101110	
241	768	0,07%	0000111100	
242	1040	0,10%	1010001101	
243	745	0,07%	0000000011	
244	952	0,09%	1000101111	
245	805	0,08%	0001010111	

246	1073	0,10%	1011100001	
247	740	0,07%	00000000101	
248	1013	0,10%	1010000000	
249	805	0,08%	0001101011	
250	1008	0,10%	1001111101	
251	796	0,08%	0001010110	
252	996	0,09%	1001110101	
253	1057	0,10%	1011000101	
254	11457	1,09%	1111010	
255	13504	1,28%	001001	
NYT (256)	0	0,00%	00000000100	

```
muutokset ioquake3:n versiolle 09/02/2014, 05:25
git-tiiviste: 90eb32c5ab01c7ee4815fd33c76fca788ff4214f
- poistettu rivi
+ lisätty rivi

--- a/code/client/cl_curl.c
+++ b/code/client/cl_curl.c
@@ -276,7 +276,7 @@ void CL_cURL_BeginDownload( const char *localName, const
char *remoteURL )
      if(!(clc.sv_allowDownload & DLF_NO_DISCONNECT) &&
            !clc.cURLDisconnected) {

-           CL_AddReliableCommand("disconnect", qtrue);
+           CL_AddReliableCommand(va("%c", rc_disconnect));
            CL_WritePacket();
            CL_WritePacket();
            CL_WritePacket();

--- a/code/client/cl_keys.c
+++ b/code/client/cl_keys.c
@@ -744,7 +744,7 @@ void Message_Key( int key ) {



-                CL_AddReliableCommand(buffer, qfalse);
+                CL_AddReliableCommand(buffer);
            }
            Key_SetCatcher( Key_GetCatcher( ) & ~KEYCATCH_MESSAGE );
            Field_Clear( &chatField );

--- a/code/client/cl_main.c
+++ b/code/client/cl_main.c
@@ -220,8 +220,8 @@ void CL_UpdateVoipIgnore(const char *idstr, qboolean ignore)
      const int id = atoi(idstr);
      if ((id >= 0) && (id < MAX_CLIENTS)) {
      clc.voipIgnore[id] = ignore;
-           CL_AddReliableCommand(va("voip %s %d",
-                 ignore ? "ignore" : "unignore", id), qfalse);
+           CL_AddReliableCommand(va("%c %s %d",
+                 clc_voip, ignore ? "ignore" : "unignore", id));
            Com_Printf("VoIP: %s ignoring player #%d\n",
                 ignore ? "Now" : "No longer", id);
            return;

@@ -283,11 +283,11 @@ void CL_Voip_f( void )
            }
      } else if (strcmp(cmd, "muteall") == 0) {
            Com_Printf("VoIP: muting incoming voice\n");
-           CL_AddReliableCommand("voip muteall", qfalse);
+           CL_AddReliableCommand(va("%cmuteall", clc_voip));
            clc.voipMuteAll = qtrue;
      } else if (strcmp(cmd, "unmuteall") == 0) {
            Com_Printf("VoIP: unmuting incoming voice\n");
-           CL_AddReliableCommand("voip unmuteall", qfalse);
```

```
+           CL_AddReliableCommand(va("%cunmuteall", clc_voip));
            clc.voipMuteAll = qfalse;
      } else {
            Com_Printf("usage: voip [un]ignore <playerID#>\n"

@@ -587,7 +587,7 @@ The given command will be transmitted to the server, and is
gauranteed to
 not have future usercmd_t executed before it is executed
 ======================
 */
-void CL_AddReliableCommand(const char *cmd, qboolean isDisconnectCmd)
+void CL_AddReliableCommand( const char *cmd, ...)
 {
      int unacknowledged = clc.reliableSequence - clc.reliableAcknowledge;

@@ -595,8 +595,8 @@ void CL_AddReliableCommand(const char *cmd, qboolean
isDisconnectCmd)
      // we must drop the connection
      // also leave one slot open for the disconnect command in this case.

-     if ((isDisconnectCmd && unacknowledged > MAX_RELIABLE_COMMANDS) ||
-         (!isDisconnectCmd && unacknowledged >= MAX_RELIABLE_COMMANDS))
+     if ((cmd[0] == rc_disconnect && unacknowledged > MAX_RELIABLE_COMMANDS) ||
+         (cmd[0] != rc_disconnect && unacknowledged >= MAX_RELIABLE_COMMANDS))
      {
            if(com_errorEntered)
                  return;

@@ -1449,7 +1449,7 @@ void CL_Disconnect( qboolean showMainMenu ) {
      // send a disconnect message to the server
      // send it a few times in case one is dropped
      if ( clc.state >= CA_CONNECTED ) {
-           CL_AddReliableCommand("disconnect", qtrue);
+           CL_AddReliableCommand(va("%c", rc_disconnect));
            CL_WritePacket();
            CL_WritePacket();
            CL_WritePacket();

@@ -1518,9 +1518,9 @@ void CL_ForwardCommandToServer( const char *string ) {
      }

      if ( Cmd_Argc() > 1 ) {
-           CL_AddReliableCommand(string, qfalse);
+           CL_AddReliableCommand(string);
      } else {
-           CL_AddReliableCommand(cmd, qfalse);
+           CL_AddReliableCommand(cmd);
      }
 }

@@ -1664,7 +1664,7 @@ void CL_ForwardToServer_f( void ) {

      // don't forward the first argument
      if ( Cmd_Argc() > 1 ) {
-           CL_AddReliableCommand(Cmd_Args(), qfalse);
+           CL_AddReliableCommand(Cmd_Args());
      }
```

```
 }

@@ -1831,9 +1831,9 @@ void CL_Rcon_f( void ) {
      message[1] = -1;
      message[2] = -1;
      message[3] = -1;
-     message[4] = 0;
+     message[4] = clc_rcon;

-     Q_strcat (message, MAX_RCON_MESSAGE, "rcon ");
+     //Q_strcat (message, MAX_RCON_MESSAGE, "rcon ");

      Q_strcat (message, MAX_RCON_MESSAGE, rcon_client_password->string);
      Q_strcat (message, MAX_RCON_MESSAGE, " ");
@@ -1869,9 +1869,9 @@ void CL_SendPureChecksums( void ) {
      char cMsg[MAX_INFO_VALUE];

      // if we are pure we need to send back a command with our referenced pk3
           checksums
-     Com_sprintf(cMsg, sizeof(cMsg), "cp %d %s", cl.serverId,
+     Com_sprintf(cMsg,   sizeof(cMsg),   "%c   %d   %s",   rc_cp,   cl.serverId,
           FS_ReferencedPakPureChecksums());

-     CL_AddReliableCommand(cMsg, qfalse);
+     CL_AddReliableCommand(cMsg);
 }

 /*
@@ -1880,7 +1880,7 @@ CL_ResetPureClientAtServer
 ==================
 */
 void CL_ResetPureClientAtServer( void ) {
-     CL_AddReliableCommand("vdr", qfalse);
+     CL_AddReliableCommand(va("%c", rc_vdr));
 }

 /*
@@ -2078,7 +2078,7 @@ void CL_DownloadsComplete( void ) {
           FS_Restart(clc.checksumFeed); // We possibly downloaded a pak,
                restart the file system to load it

           // inform the server so we get new gamestate info
-          CL_AddReliableCommand("donedl", qfalse);
+          CL_AddReliableCommand(va("%c", rc_doneDl));

           // by sending the donedl command we request a new gamestate

@@ -2145,7 +2145,7 @@ void CL_BeginDownload( const char *localName, const char
*remoteName ) {
      clc.downloadBlock = 0; // Starting new file
      clc.downloadCount = 0;

-     CL_AddReliableCommand(va("download %s", remoteName), qfalse);
+     CL_AddReliableCommand(va("%c %s", rc_download, remoteName), qfalse);
 }
```

```
 /*
@@ -2334,7 +2334,7 @@ void CL_CheckForResend( void ) {
             // with a meaningful message
-            Com_sprintf(data, sizeof(data), "getchallenge %d %s", clc.challenge,
                   com_gamename->string);
+            Com_sprintf(data, sizeof(data), "%c%d %s", clc_getChallenge,
                   clc.challenge, com_gamename->string);

             NET_OutOfBandPrint(NS_CLIENT, clc.serverAddress, "%s", data);
             break;

@@ -2357,16 +2357,15 @@ void CL_CheckForResend( void ) {
             Info_SetValueForKey( info, "qport", va("%i", port ) );
             Info_SetValueForKey( info, "challenge", va("%i", clc.challenge ) );

-            strcpy(data, "connect ");
-    // TTimo adding " " around the userinfo string to avoid truncated userinfo
             on the server
-    //   (Com_TokenizeString tokenizes around spaces)
-    data[8] = '"';
-
-            for(i=0;i<strlen(info);i++) {
-                  data[9+i] = info[i];    // + (clc.challenge)&0x3;
-            }
-    data[9+i] = '"';
-    data[10+i] = 0;
+    data[0] = clc_connect | ADAPTIVE_HUFFMAN;
+    // TTimo adding " " around the userinfo string to avoid truncated
      userinfo on the server
+    //   (Com_TokenizeString tokenizes around spaces)
+    data[1] = '"';
+    for(i=0;i<strlen(info);i++) {
+         data[2+i] = info[i];   // + (clc.challenge)&0x3;
+    }
+    data[2+i] = '"';
+    data[3+i] = 0;

@@ -2565,26 +2564,26 @@ Responses to broadcasts, etc
 =================
 */
 void CL_ConnectionlessPacket( netadr_t from, msg_t *msg ) {
-     char  *s;
-     char  *c;
-     int challenge = 0;
+     char *args, *strver, *temp;
+     int challenge, cmd, ver = 0;

      MSG_BeginReadingOOB( msg );
      MSG_ReadLong( msg );    // skip the -1
-
-     s = MSG_ReadStringLine( msg );
-
-     Cmd_TokenizeString( s );
-
-     c = Cmd_Argv(0);
-
-     Com_DPrintf ("CL packet %s: %s\n", NET_AdrToStringwPort(from), c);
```

```
-
-	// challenge from the server we are connecting to
-	if (!Q_stricmp(c, "challengeResponse"))
-	{
-		char *strver;
-		int ver;
+	cmd = MSG_ReadByte( msg );	// read cmd
+
+	// check for adaptive huffman coding
+	if (cmd & ADAPTIVE_HUFFMAN) {
+		Huff_Decompress( msg, msg->readcount ); // decode using adaptive
		huffman algorithm
+		cmd &= ~ADAPTIVE_HUFFMAN; // remove huffman bit
+	}
+
+	Com_DPrintf ("CL packet %s: %d\n", NET_AdrToStringwPort(from), cmd);
+
+	switch ( cmd ) {
+		// challenge from the server we are connecting to
+		case svc_challengeResponse:
+			args = MSG_ReadStringLine( msg );
+			Cmd_TokenizeString( args );

@@ -2592,12 +2591,12 @@ void CL_ConnectionlessPacket( netadr_t from, msg_t
*msg ) {
			return;
		}

-		c = Cmd_Argv(2);
-		if(*c)
-			challenge = atoi(c);
+		temp = Cmd_Argv(1);
+		if(*args)
+			challenge = atoi(temp);

-		strver = Cmd_Argv(3);
-		if(*strver)
+			strver = Cmd_Argv(2);
+			if(*strver)
		{
			ver = atoi(strver);

@@ -2634,7 +2633,7 @@ void CL_ConnectionlessPacket( netadr_t from, msg_t *msg )
{
		// Check whether we have a matching client challenge to prevent
		// connection hi-jacking.

-		if(!*c || challenge != clc.challenge)
+		if(!*temp || challenge != clc.challenge)
		{

@@ -2644,7 +2643,7 @@ void CL_ConnectionlessPacket( netadr_t from, msg_t *msg )
 #endif
		{
-		if(!*c || challenge != clc.challenge)
+		if(!*temp || challenge != clc.challenge)
		{
```

```
                Com_Printf("Bad challenge for challengeResponse. Ignored.\n");
                return;

@@ -2652,7 +2651,7 @@ void CL_ConnectionlessPacket( netadr_t from, msg_t *msg )
                // start sending challenge response instead of challenge request
                packets
-               clc.challenge = atoi(Cmd_Argv(1));
+               clc.challenge = atoi(Cmd_Argv(0));
                clc.state = CA_CHALLENGING;

@@ -2661,32 +2660,28 @@ void CL_ConnectionlessPacket( netadr_t from, msg_t
*msg ) {
                Com_DPrintf ("challengeResponse: %d\n", clc.challenge);
-               return;
-       }
-
-       // server connection
-       if ( !Q_stricmp(c, "connectResponse") ) {
-               if ( clc.state >= CA_CONNECTED ) {
-                       Com_Printf ("Dup connect received. Ignored.\n");
-                       return;
-               }
-               if ( clc.state != CA_CHALLENGING ) {
+               break;
+
+               // server connection
+               case svc_connectResponse:
+                       if ( clc.state >= CA_CONNECTED ) {
+                           Com_Printf ("Dup connect received.  Ignored.\n");
+                           break;
+                       }
+                       if ( clc.state != CA_CHALLENGING ) {
                                Com_Printf ("connectResponse packet while not
                                  connecting. Ignored.\n");
-                               return;
+                               break;
                        }
-               if ( !NET_CompareAdr( from, clc.serverAddress ) ) {
+                       if ( !NET_CompareAdr( from, clc.serverAddress ) ) {
                                Com_Printf( "connectResponse from wrong address.
                                   Ignored.\n" );
-                               return;
+                               break;
                        }

 #ifdef LEGACY_PROTOCOL
                if(!clc.compat)
 #endif
                {
-                       c = Cmd_Argv(1);
+                       temp = Cmd_Argv(0);

-                       if(*c)
-                               challenge = atoi(c);
+                       if(*temp)
+                               challenge = atoi(temp);
                        else
```

```
                {

@@ -2710,62 +2705,48 @@ void CL_ConnectionlessPacket( netadr_t from, msg_t
*msg ) {

            clc.lastPacketSentTime = -9999;     // send first packet immediately
-           return;
-     }
-
-     // server responding to an info broadcast
-     if ( !Q_stricmp(c, "infoResponse") ) {
-           CL_ServerInfoPacket( from, msg );
-           return;
-     }
-
-     // server responding to a get playerlist
-     if ( !Q_stricmp(c, "statusResponse") ) {
-           CL_ServerStatusResponse( from, msg );
-           return;
-     }
-
-     // echo request from server
-     if ( !Q_stricmp(c, "echo") ) {
-           NET_OutOfBandPrint( NS_CLIENT, from, "%s", Cmd_Argv(1) );
-           return;
-     }
-
-     // cd check
-     if ( !Q_stricmp(c, "keyAuthorize") ) {
-           // we don't use these now, so dump them on the floor
-           return;
-     }
-
-     // global MOTD from id
-     if ( !Q_stricmp(c, "motd") ) {
-           CL_MotdPacket( from );
-           return;
-     }
-
-     // echo request from server
-     if(!Q_stricmp(c, "print")){
-           s = MSG_ReadString( msg );
-
-           Q_strncpyz( clc.serverMessage, s, sizeof( clc.serverMessage ) );
-           Com_Printf( "%s", s );
-
-           return;
-     }
-
-     // list of servers sent back by a master server (classic)
-     if ( !Q_strncmp(c, "getserversResponse", 18) ) {
-           CL_ServersResponsePacket( &from, msg, qfalse );
-           return;
-     }
-
-     // list of servers sent back by a master server (extended)
-     if ( !Q_strncmp(c, "getserversExtResponse", 21) ) {
```

```
-            CL_ServersResponsePacket( &from, msg, qtrue );
-            return;
-      }
+            break;

-      Com_DPrintf ("Unknown connectionless packet command.\n");
+       // server responding to an info broadcast
+       case svc_infoResponse:
+          CL_ServerInfoPacket( from, msg );
+          break;
+       // server responding to a get playerlist
+       case svc_statusResponse:
+          CL_ServerStatusResponse( from, msg );
+          break;
+       // echo request from server
+       case svc_echo:
+          NET_OutOfBandPrint( NS_CLIENT, from, "%s", MSG_ReadString( msg ) );
+          break;
+       // cd check
+       case svc_keyAuthorize:
+          // we don't use these now, so dump them on the floor
+          break;
+       // global MOTD from id
+       case svc_motd:
+          CL_MotdPacket( from );
+          break;
+       // echo request from server
+       case svc_print:
+          args = MSG_ReadString( msg );
+          Q_strncpyz( clc.serverMessage, args, sizeof( clc.serverMessage ) );
+          Com_Printf( "%s", args );
+          break;
+      // list of servers sent back by a master server (classic)
+      case svc_getServersResponse:
+         CL_ServersResponsePacket( &from, msg, qfalse );
+         break;
+      // list of servers sent back by a master server (extended)
+      case svc_getServersExtResponse:
+         CL_ServersResponsePacket( &from, msg, qtrue );
+         break;
+      default:
+         Com_DPrintf ("Unknown connectionless packet command.\n");
+         break;
+      }
  }

@@ -2890,7 +2871,7 @@ void CL_CheckUserinfo( void ) {
       if(cvar_modifiedFlags & CVAR_USERINFO)
       {
             cvar_modifiedFlags &= ~CVAR_USERINFO;
-            CL_AddReliableCommand(va("userinfo \"%s\"",
+            CL_AddReliableCommand(va("%c \"%s\"", rc_userinfo,
                   Cvar_InfoString( CVAR_USERINFO ) ), qfalse);
       }
```

```
@@ -3928,7 +3909,7 @@ int CL_ServerStatus( char *serverAddress, char
*serverStatusString, int maxLen )
            serverStatus->startTime = Com_Milliseconds();
-           NET_OutOfBandPrint( NS_CLIENT, to, "getstatus" );
+           NET_OutOfBandPrint( NS_CLIENT, to, "%c", clc_getStatus );
            return qfalse;

@@ -3940,7 +3921,7 @@ int CL_ServerStatus( char *serverAddress, char
*serverStatusString, int maxLen )
            serverStatus->time = 0;
-           NET_OutOfBandPrint( NS_CLIENT, to, "getstatus" );
+           NET_OutOfBandPrint( NS_CLIENT, to, "%c", clc_getStatus );
            return qfalse;

@@ -4045,7 +4026,7 @@ CL_LocalServers_f
 void CL_LocalServers_f( void ) {
-     char        *message;
+     char        message[MAX_MSGLEN];
      int         i, j;
      netadr_t    to;

@@ -4065,7 +4046,7 @@ void CL_LocalServers_f( void ) {
      // can use that to prevent spoofed server responses from invalid ip
-     message = "\377\377\377\377getinfo xxx";
+     Com_sprintf (message, sizeof(message), "\xff\xff\xff\xff%cxxx",
           clc_getInfo);

      // send each message twice in case one is dropped
      for ( i = 0 ; i < 2 ; i++ ) {

@@ -4360,7 +4341,7 @@ void CL_Ping_f( void ) {

      CL_SetServerInfoByAddress(pingptr->adr, NULL, 0);

-     NET_OutOfBandPrint( NS_CLIENT, to, "getinfo xxx" );
+     NET_OutOfBandPrint( NS_CLIENT, to, "%cxxx", clc_getInfo );
 }

 /*

@@ -4428,7 +4409,7 @@ qboolean CL_UpdateVisiblePings_f(int source) {
      cl_pinglist[j].time = 0;
-     NET_OutOfBandPrint( NS_CLIENT, cl_pinglist[j].adr, "getinfo xxx" );
+     NET_OutOfBandPrint( NS_CLIENT, cl_pinglist[j].adr, "%cxxx", clc_getInfo );
      slots++;
      }

@@ -4516,7 +4497,7 @@ void CL_ServerStatus_f(void) {
                  return;
      }

-     NET_OutOfBandPrint( NS_CLIENT, *toptr, "getstatus" );
+     NET_OutOfBandPrint( NS_CLIENT, *toptr, "%c", clc_getStatus );

      serverStatus = CL_GetServerStatus( *toptr );
```

```
--- a/code/client/cl_parse.c
+++ b/code/client/cl_parse.c

@@ -568,7 +568,7 @@ void CL_ParseDownload ( msg_t *msg ) {
            Com_Printf("Server   sending   download,   but   no   download   was
requested\n");
-           CL_AddReliableCommand("stopdl", qfalse);
+           CL_AddReliableCommand(va("%c", rc_stopDl));
            return;

@@ -611,7 +611,7 @@ void CL_ParseDownload ( msg_t *msg ) {

            Com_Printf( "Could not create %s\n", clc.downloadTempName );
-           CL_AddReliableCommand("stopdl", qfalse);
+           CL_AddReliableCommand("%c", va("%c", rc_stopDl));
            CL_NextDownload();
            return;

@@ -620,7 +620,7 @@ void CL_ParseDownload ( msg_t *msg ) {
      if (size)
            FS_Write( data, size, clc.download );

-     CL_AddReliableCommand(va("nextdl %d", clc.downloadBlock), qfalse);
+     CL_AddReliableCommand(va("%c %d", rc_nextDl, clc.downloadBlock), qfalse);
      clc.downloadBlock++;

@@ -898,10 +898,10 @@ void CL_ParseServerMessage( msg_t *msg ) {
            }

            if ( cl_shownet->integer >= 2 ) {
-                 if ( (cmd < 0) || (!svc_strings[cmd]) ) {
+                 if ( (cmd < 0) || (cmd > svc_count) ) {
                        Com_Printf( "%3i:BAD CMD %i\n", msg->readcount-1, cmd );
                  } else {
-                       SHOWNET( msg, svc_strings[cmd] );
+                               SHOWNET( msg, va("%d", cmd) );
                  }
            }
```

```
--- a/code/client/client.h
+++ b/code/client/client.h
@@ -456,7 +456,7 @@ extern    cvar_t      *cl_voip;
 //

 void CL_Init (void);
-void CL_AddReliableCommand(const char *cmd, qboolean isDisconnectCmd);
+void CL_AddReliableCommand( const char *cmd, ...);

 void CL_StartHunkUsers( qboolean rendererOnly );
```

```
--- a/code/qcommon/huffman.c
+++ b/code/qcommon/huffman.c
@@ -380,6 +380,7 @@ void Huff_Decompress(msg_t *mbuf, int offset) {
      /* Increment node */
      }
      mbuf->cursize = cch + offset;
+     mbuf->decoded = qtrue;
      Com_Memcpy(mbuf->data + offset, seq, cch);
 }

@@ -420,6 +421,7 @@ void Huff_Compress(msg_t *mbuf, int offset) {
      // next byte
      mbuf->cursize = (bloc>>3) + offset;
+     mbuf->decoded = qfalse;
      Com_Memcpy(mbuf->data+offset, seq, (bloc>>3));
 }

--- a/code/qcommon/msg.c
+++ b/code/qcommon/msg.c
@@ -58,6 +58,7 @@ void MSG_InitOOB( msg_t *buf, byte *data, int length ) {
      buf->oob = qtrue;
+     buf->decoded = qtrue;
 }

 void MSG_Clear( msg_t *buf ) {
@@ -75,12 +76,14 @@ void MSG_BeginReading( msg_t *msg ) {
      msg->oob = qfalse;
+     msg->decoded = qfalse;
 }

 void MSG_BeginReadingOOB( msg_t *msg ) {
      msg->oob = qtrue;
+     msg->decoded = qfalse;
 }

 void MSG_Copy(msg_t *buf, byte *data, int length, msg_t *src)
@@ -139,7 +142,7 @@ void MSG_WriteBits( msg_t *msg, int value, int bits ) {
      if ( bits < 0 ) {
           bits = -bits;
      }
-     if (msg->oob) {
+     if (msg->oob || msg->decoded) {
           if(bits==8)
           {
                msg->data[msg->cursize] = value;

@@ -202,7 +205,7 @@ int MSG_ReadBits( msg_t *msg, int bits ) {
           sgn = qfalse;
      }
-     if (msg->oob) {
+     if (msg->oob || msg->decoded) {
          if(bits==8)
          {
               value = msg->data[msg->readcount];

--- a/code/qcommon/net_chan.c
+++ b/code/qcommon/net_chan.c
```

```
@@ -591,9 +591,9 @@ void QDECL NET_OutOfBandPrint( netsrc_t sock, netadr_t adr,
const char *format,

 /*
 ===============
-NET_OutOfBandPrint
+NET_OutOfBandData

-Sends a data message in an out-of-band datagram (only used for "connect")
+Sends a data message in an out-of-band datagram, uses adaptive Huffman coding
if ADAPTIVE_HUFFMAN bit is set
 ================
 */
 void QDECL NET_OutOfBandData( netsrc_t sock, netadr_t adr, byte *format, int
len ) {
@@ -613,7 +613,10 @@ void QDECL NET_OutOfBandData( netsrc_t sock, netadr_t adr,
byte *format, int len

       mbuf.data = string;
       mbuf.cursize = len+4;
-      Huff_Compress( &mbuf, 12);
+
+      // check if this needs to be coded using adaptive Huffman algorithm
+      if (mbuf.data[4] & ADAPTIVE_HUFFMAN)
+            Huff_Compress( &mbuf, 5); // header and cmd are sent plaintext.
                  assumes one-byte cmd (protocol 72)
       // send the datagram
       NET_SendPacket( sock, mbuf.cursize, mbuf.data, adr );
 }

--- a/code/qcommon/q_shared.h
+++ b/code/qcommon/q_shared.h
@@ -1103,6 +1102,86 @@ typedef struct {
       int               dataCount;
 } gameState_t;

+// use adaptive Huffman coding, used with command byte
+#define ADAPTIVE_HUFFMAN        128
+
+//
+// reliable commands. can be client to server or vice versa. protocol 72 only.
+// these bytes replace the old string-based commands. reuse only safe values
+//
+
+enum rel_ops_e {
+      rc_userinfo,
+      rc_disconnect,
+      rc_cp,
+      rc_vdr,
+      rc_download,
+      rc_nextDl,
+      rc_stopDl,
+      rc_doneDl,
+      rc_chat,
+      rc_tchat,
+      rc_scores,
+      rc_count,
```

```
+};
+
+
+//
+// server to client
+//
+enum svc_ops_e {
+     svc_bad = rc_count,
+     svc_nop,
+     svc_gamestate,
+     svc_configstring,       // [short] [string] only in gamestate messages
+     svc_baseline,           // only in gamestate messages
+     svc_serverCommand,      // [string] to be executed by client game module
+     svc_download,           // [short] size [size bytes]
+     svc_snapshot,
+     svc_EOF,
+
+     // new commands, supported only by ioquake3 protocol but not legacy
+     svc_voip,     // not wrapped in USE_VOIP, so this value is reserved.
+
+// protocol 72
+     svc_challengeResponse,
+     svc_connectResponse,
+     svc_infoResponse,
+     svc_statusResponse,
+     svc_echo,
+     svc_keyAuthorize,
+     svc_motd,
+     svc_print,
+     svc_centerprint,
+     svc_getServersResponse,
+     svc_getServersExtResponse,
+      svc_count,              // if more commands are added, use values larges
than 128 and make necessary changes
+};
+
+//
+// client to server
+//
+enum clc_ops_e {
+     clc_bad = svc_challengeResponse,
+     clc_nop,
+     clc_move,               // [[usercmd_t]
+     clc_moveNoDelta,        // [[usercmd_t]
+     clc_clientCommand,      // [string] message
+     clc_EOF,
+
+     // new commands, supported only by ioquake3 protocol but not legacy
+     clc_voip,   // not wrapped in USE_VOIP, so this value is reserved.
+
+// protocol 72
+     clc_getChallenge,
+     clc_connect,
+     clc_getInfo,
+     clc_getStatus,
+     clc_ipAuthorize,
+     clc_rcon,
```

```
+     clc_count,
+};
+

--- a/code/qcommon/qcommon.h
+++ b/code/qcommon/qcommon.h
@@ -43,6 +43,7 @@ typedef struct {
      qboolean    oob;          // set to true if the buffer size failed (with
                                     allowoverflow set)
+     qboolean    decoded;      // set to true if the message is already decoded
      byte  *data;
      int         maxsize;
      int         cursize;

@@ -251,7 +252,7 @@ PROTOCOL
 ==============================================================
 */

-#define    PROTOCOL_VERSION  71
+#define    PROTOCOL_VERSION  72
 #define PROTOCOL_LEGACY_VERSION    68
 // 1.31 - 67

@@ -283,42 +284,6 @@ extern int demo_protocols[];
// PORT_SERVER so a single machine can
// run multiple servers

-
-// the svc_strings[] array in cl_parse.c should mirror this
-//
-// server to client
-//
-enum svc_ops_e {
-     svc_bad,
-     svc_nop,
-     svc_gamestate,
-     svc_configstring,            //  [short]  [string]  only  in  gamestate
messages
-     svc_baseline,                  // only in gamestate messages
-     svc_serverCommand,             // [string] to be executed by client
game module
-     svc_download,                  // [short] size [size bytes]
-     svc_snapshot,
-     svc_EOF,
-
-// new commands, supported only by ioquake3 protocol but not legacy
-     svc_voip,     // not wrapped in USE_VOIP, so this value is reserved.
-};
-
-
-//
-// client to server
-//
-enum clc_ops_e {
-     clc_bad,
-     clc_nop,
-     clc_move,                  // [[usercmd_t]
```

```
-     clc_moveNoDelta,        // [[usercmd_t]
-     clc_clientCommand,            // [string] message
-     clc_EOF,
-
-// new commands, supported only by ioquake3 protocol but not legacy
-     clc_voip,   // not wrapped in USE_VOIP, so this value is reserved.
-};
-

--- a/code/server/sv_client.c
+++ b/code/server/sv_client.c
@@ -81,7 +81,7 @@ void SV_GetChallenge(netadr_t from)
            return;
      }

-     gameName = Cmd_Argv(2);
+     gameName = Cmd_Argv(1);

 #ifdef LEGACY_PROTOCOL
      // gamename is optional for legacy protocol

@@ -94,7 +94,7 @@ void SV_GetChallenge(netadr_t from)
      if (gameMismatch)
      {
-           NET_OutOfBandPrint(NS_SERVER, from, "print\nGame mismatch: This is a
                %s server\n",
+           NET_OutOfBandPrint(NS_SERVER, from, "%c\nGame mismatch: This is a %s
                server\n", svc_print, com_gamename->string);
            return;
      }
@@ -104,7 +104,7 @@ void SV_GetChallenge(netadr_t from)

      // see if we already have a challenge for this ip
      challenge = &svs.challenges[0];
-     clientChallenge = atoi(Cmd_Argv(1));
+     clientChallenge = atoi(Cmd_Argv(0));

      for(i = 0 ; i < MAX_CHALLENGES ; i++, challenge++)
      {
@@ -199,7 +199,7 @@ void SV_GetChallenge(netadr_t from)
 #endif

      challenge->pingTime = svs.time;
-     NET_OutOfBandPrint(NS_SERVER, challenge->adr, "challengeResponse %d %d
      %d",
+     NET_OutOfBandPrint(NS_SERVER, challenge->adr, "%c%d %d %d",
            svc_challengeResponse,
            challenge->challenge, clientChallenge, com_protocol->integer);
 }

@@ -246,7 +246,7 @@ void SV_AuthorizeIpPacket( netadr_t from ) {
      if ( !Q_stricmp( s, "demo" ) ) {
            // they are a demo client trying to connect to a real server
-           NET_OutOfBandPrint( NS_SERVER, challengeptr->adr, "print\nServer is
                not a demo server\n" );
+           NET_OutOfBandPrint( NS_SERVER, challengeptr->adr, "%cServer is not a
                demo server\n", svc_print );
```

```
            // clear the challenge record so it won't timeout and let them
                through

@@ -258,9 +258,9 @@ void SV_AuthorizeIpPacket( netadr_t from ) {
      }
      if ( !Q_stricmp( s, "unknown" ) ) {
            if (!r) {
-     NET_OutOfBandPrint( NS_SERVER, challengeptr->adr, "print\nAwaiting CD key
            authorization\n" );
+     NET_OutOfBandPrint( NS_SERVER, challengeptr->adr, "%cAwaiting CD key
            authorization\n", svc_print );
            } else {
-     NET_OutOfBandPrint( NS_SERVER, challengeptr->adr, "print\n%s\n", r);
+     NET_OutOfBandPrint( NS_SERVER, challengeptr->adr, "%c%s\n", svc_print, r);
            }
      // clear the challenge record so it won't timeout and let them through
            Com_Memset( challengeptr, 0, sizeof( *challengeptr ) );

@@ -269,9 +269,9 @@ void SV_AuthorizeIpPacket( netadr_t from ) {

      // authorization failed
      if (!r) {
-           NET_OutOfBandPrint( NS_SERVER, challengeptr->adr, "print\nSomeone is
                using this CD Key\n" );
+           NET_OutOfBandPrint( NS_SERVER, challengeptr->adr, "%cSomeone is
                using this CD Key\n", svc_print );
      } else {
-           NET_OutOfBandPrint( NS_SERVER, challengeptr->adr, "print\n%s\n",
            r );
+           NET_OutOfBandPrint( NS_SERVER, challengeptr->adr, "%c%s\n",
                svc_print, r );
      }

      // clear the challenge record so it won't timeout and let them through
@@ -345,11 +345,11 @@ void SV_DirectConnect( netadr_t from ) {
      // Check whether this client is banned.
      if(SV_IsBanned(&from, qfalse))
      {
-           NET_OutOfBandPrint(NS_SERVER, from, "print\nYou are banned from this
                server.\n");
+           NET_OutOfBandPrint(NS_SERVER, from, "%cYou are banned from this
                server.\n", svc_print);
            return;
      }

-     Q_strncpyz( userinfo, Cmd_Argv(1), sizeof(userinfo) );
+     Q_strncpyz( userinfo, Cmd_Argv(0), sizeof(userinfo) );

      version = atoi(Info_ValueForKey(userinfo, "protocol"));

@@ -361,8 +361,8 @@ void SV_DirectConnect( netadr_t from ) {
      {
            if(version != com_protocol->integer)
            {
-                 NET_OutOfBandPrint(NS_SERVER, from, "print\nServer uses
                      protocol version %i "
-                       "(yours is %i).\n", com_protocol->integer, version);
```

```
+                 NET_OutOfBandPrint(NS_SERVER, from, "%cServer uses protocol
                        version %i "
+                       "(yours is %i).\n", svc_print, com_protocol->integer,
                        version);
                  Com_DPrintf("    rejected connect from version %i\n",
                        version);
                  return;

@@ -395,8 +395,8 @@ void SV_DirectConnect( netadr_t from ) {
            ip = (char *)NET_AdrToString( from );
      if( ( strlen( ip ) + strlen( userinfo ) + 4 ) >= MAX_INFO_STRING ) {
            NET_OutOfBandPrint( NS_SERVER, from,
-                 "print\nUserinfo string length exceeded.  "
-                 "Try removing setu cvars from your config.\n" );
+                 "%cUserinfo string length exceeded.  "
+                 "Try removing setu cvars from your config.\n", svc_print );
            return;
      }

@@ -418,7 +418,7 @@ void SV_DirectConnect( netadr_t from ) {

            if (i == MAX_CHALLENGES)
            {
-                 NET_OutOfBandPrint( NS_SERVER, from, "print\nNo or bad
                        challenge for your address.\n" );
+                 NET_OutOfBandPrint( NS_SERVER, from, "%cNo or bad challenge
                        for your address.\n", svc_print );
                  return;
            }

@@ -435,13 +435,13 @@ void SV_DirectConnect( netadr_t from ) {
            // never reject a LAN client based on ping
            if ( !Sys_IsLANAddress( from ) ) {
                  if ( sv_minPing->value && ping < sv_minPing->value ) {
-                       NET_OutOfBandPrint( NS_SERVER, from, "print\nServer is
                              for high pings only\n" );
+                       NET_OutOfBandPrint( NS_SERVER, from, "%cServer is for
                              high pings only\n", svc_print );
                        Com_DPrintf ("Client %i rejected on a too low ping\n",
                              i);
                        challengeptr->wasrefused = qtrue;
                        return;
                  }
                  if ( sv_maxPing->value && ping > sv_maxPing->value ) {
-                       NET_OutOfBandPrint( NS_SERVER, from, "print\nServer is
                              for low pings only\n" );
+                       NET_OutOfBandPrint( NS_SERVER, from, "%cServer is for
                              low pings only\n", svc_print );
                        Com_DPrintf ("Client %i rejected on a too high ping\n",
                              i);
                        challengeptr->wasrefused = qtrue;
                        return;

@@ -524,7 +524,7 @@ void SV_DirectConnect( netadr_t from ) {
            else {
-                 NET_OutOfBandPrint( NS_SERVER, from, "print\nServer is
                        full.\n" );
```

```
+                NET_OutOfBandPrint( NS_SERVER, from, "%cServer is full.\n",
                       svc_print );
                 Com_DPrintf ("Rejected a connection.\n");

@@ -565,7 +565,7 @@ gotnewcl:
             char *str = VM_ExplicitArgPtr( gvm, denied );

-            NET_OutOfBandPrint( NS_SERVER, from, "print\n%s\n", str );
+            NET_OutOfBandPrint( NS_SERVER, from, "%c%s\n", svc_print, str );
             Com_DPrintf ("Game rejected a connection: %s.\n", str);

@@ -573,7 +573,7 @@ gotnewcl:
       SV_UserinfoChanged( newcl );

       // send the connect packet to the client
-      NET_OutOfBandPrint(NS_SERVER, from, "connectResponse %d", challenge);
+      NET_OutOfBandPrint(NS_SERVER, from, "%c%d", svc_connectResponse,
       challenge);

       Com_DPrintf( "Going from CS_FREE to CS_CONNECTED for %s\n", newcl->name );

@@ -1245,8 +1245,8 @@ static void SV_VerifyPaks_f( client_t *cl ) {

             nClientPaks = Cmd_Argc();

-            // start at arg 2 ( skip serverId cl_paks )
-            nCurArg = 1;
+            // start at arg 1 ( skip serverId cl_paks )
+            nCurArg = 0;

             pArg = Cmd_Argv(nCurArg++);
             if(!pArg) {

@@ -1533,25 +1533,25 @@ static void SV_Voip_f( client_t *cl ) {
 typedef struct {
-      char  *name;
+      char  name;
       void  (*func)( client_t *cl );
 } ucmd_t;

 static ucmd_t ucmds[] = {
-      {"userinfo", SV_UpdateUserinfo_f},
-      {"disconnect", SV_Disconnect_f},
-      {"cp", SV_VerifyPaks_f},
-      {"vdr", SV_ResetPureClient_f},
-      {"download", SV_BeginDownload_f},
-      {"nextdl", SV_NextDownload_f},
-      {"stopdl", SV_StopDownload_f},
-      {"donedl", SV_DoneDownload_f},
+      {rc_userinfo, SV_UpdateUserinfo_f},
+      {rc_disconnect, SV_Disconnect_f},
+      {rc_cp, SV_VerifyPaks_f},
+      {rc_vdr, SV_ResetPureClient_f},
+      {rc_download, SV_BeginDownload_f},
+      {rc_nextDl, SV_NextDownload_f},
+      {rc_stopDl, SV_StopDownload_f},
+      {rc_doneDl, SV_DoneDownload_f},
```

```
 #ifdef USE_VOIP
-     {"voip", SV_Voip_f},
+     {clc_voip, SV_Voip_f},
 #endif

-     {NULL, NULL}
+     {0, NULL}
 };

 /*

@@ -1568,8 +1568,8 @@ void SV_ExecuteClientCommand( client_t *cl, const char *s,
qboolean clientOK ) {
      Cmd_TokenizeString( s );

      // see if it is a server level command
-     for (u=ucmds ; u->name ; u++) {
-          if (!strcmp (Cmd_Argv(0), u->name) ) {
+     for (u=ucmds ; u->func ; u++) {
+          if ( s[0] == u->name ) {
                u->func( cl );
                bProcessed = qtrue;
                break;

@@ -1605,7 +1605,7 @@ static qboolean SV_ClientCommand( client_t *cl, msg_t *msg
) {
          return qtrue;
      }

-     Com_DPrintf( "clientCommand: %s : %i : %s\n", cl->name, seq, s );
+     Com_DPrintf( "clientCommand: %s : %i : %d\n", cl->name, seq, s[0] );

      // drop the connection if we have somehow lost commands
      if ( seq > cl->lastClientCommand + 1 ) {

--- a/code/server/sv_main.c
+++ b/code/server/sv_main.c
@@ -584,7 +584,7 @@ static void SVC_Status( netadr_t from ) {
          }
      }

-     NET_OutOfBandPrint( NS_SERVER, from, "statusResponse\n%s\n%s", infostring,
          status );
+     NET_OutOfBandPrint( NS_SERVER, from, "%c%s\n%s", svc_statusResponse,
          infostring, status );
 }

 /*

@@ -597,6 +597,7 @@ if a user is interested in a server to do a full status
 */
 void SVC_Info( netadr_t from ) {
      int   i, count, humans;
+     char  *finalString;
      char  *gamedir;
      char  infostring[MAX_INFO_STRING];
```

```
@@ -681,7 +682,8 @@ void SVC_Info( netadr_t from ) {
            Info_SetValueForKey( infostring, "game", gamedir );
      }

-   NET_OutOfBandPrint( NS_SERVER, from, "infoResponse\n%s", infostring );
+   finalString = va("%c%s", (svc_infoResponse | ADAPTIVE_HUFFMAN), infostring);
+   NET_OutOfBandData( NS_SERVER, from, (byte*)finalString,
+           strlen(finalString) );
 }

 /*

@@ -691,7 +693,7 @@ SVC_FlushRedirect
 ================
 */
 static void SV_FlushRedirect( char *outputbuf ) {
-NET_OutOfBandPrint( NS_SERVER, svs.redirectAddress, "print\n%s", outputbuf );
+NET_OutOfBandPrint( NS_SERVER, svs.redirectAddress, "%c%s", svc_print,
      outputbuf );
 }

 /*

@@ -720,7 +722,7 @@ static void SVC_RemoteCommand( netadr_t from, msg_t *msg ) {
      }

      if ( !strlen( sv_rconPassword->string ) ||
-           strcmp (Cmd_Argv(1), sv_rconPassword->string) ) {
+           strcmp (Cmd_Argv(0), sv_rconPassword->string) ) {
+           static leakyBucket_t bucket;

            // Make DoS via rcon impractical

@@ -780,44 +782,46 @@ connectionless packets.
 ==================
 */
 static void SV_ConnectionlessPacket( netadr_t from, msg_t *msg ) {
-     char  *s;
-     char  *c;
+     char    *args;
+     int     cmd;

      MSG_BeginReadingOOB( msg );
      MSG_ReadLong( msg );          // skip the -1 marker
+     cmd = MSG_ReadByte( msg );  // read cmd

-     if (!Q_strncmp("connect", (char *) &msg->data[4], 7)) {
-           Huff_Decompress(msg, 12);
-     }
-
-     s = MSG_ReadStringLine( msg );
-     Cmd_TokenizeString( s );
-
-     c = Cmd_Argv(0);
-     Com_DPrintf ("SV packet %s : %s\n", NET_AdrToString(from), c);
```

```
-
-     if (!Q_stricmp(c, "getstatus")) {
-           SVC_Status( from );
-   } else if (!Q_stricmp(c, "getinfo")) {
-           SVC_Info( from );
-     } else if (!Q_stricmp(c, "getchallenge")) {
-           SV_GetChallenge(from);
-     } else if (!Q_stricmp(c, "connect")) {
-           SV_DirectConnect( from );
-#ifndef STANDALONE
-     } else if (!Q_stricmp(c, "ipAuthorize")) {
-           SV_AuthorizeIpPacket( from );
-#endif
-     } else if (!Q_stricmp(c, "rcon")) {
-           SVC_RemoteCommand( from, msg );
-     } else if (!Q_stricmp(c, "disconnect")) {
-           // if a client starts up a local server, we may see some spurious
-           // server disconnect messages when their new server sees our final
-           // sequenced messages to the old client
-     } else {
-           Com_DPrintf ("bad connectionless packet from %s:\n%s\n",
-                 NET_AdrToString (from), s);
+     if (cmd & ADAPTIVE_HUFFMAN) {
+        Huff_Decompress( msg, msg->readcount ); // decode using adaptive
                huffman algorithm
+        cmd &= ~ADAPTIVE_HUFFMAN; // remove ADAPTIVE_HUFFMAN bit
      }
+        args = MSG_ReadStringLine( msg );
+        Cmd_TokenizeString( args );
+        Com_DPrintf ("SV packet %s : %d\n", NET_AdrToString(from), cmd);
+
+        switch ( cmd ) {
+             case clc_getStatus:
+                 SVC_Status( from );
+                 break;
+             case clc_getInfo:
+                 SVC_Info( from );
+                 break;
+             case clc_getChallenge:
+                 SV_GetChallenge( from );
+                 break;
+             case clc_connect:
+                 SV_DirectConnect( from );
+                 break;
+             case clc_rcon:
+                 SVC_RemoteCommand( from, msg );
+                 break;
+             case rc_disconnect:
+             // if a client starts up a local server, we may see some spurious
+             // server disconnect messages when their new server sees our final
+             // sequenced messages to the old client
+                 break;
+             default:
+                  Com_DPrintf ("bad connectionless packet from %s:\n%d %s\n",
                       NET_AdrToString (from), cmd, args);
+                 break;
+         }
```